RT–11 System Utilities Manual

AA–M239A–TC

March 1983

This document describes how to use the RT–11 operating system. The manual provides the information required to perform ordinary tasks such as program development, program execution, and file maintenance by using RT–11 system utility programs.

Operating System: RT–11 Version 5.0

Software: RT–11 Version 5.0

To order additional documents from within DIGITAL, contact the Software Distribution Center, Northboro, Massachusetts 01532.

To order additional documents from outside DIGITAL, refer to the instructions at the back of this document.

digital equipment corporation · maynard, massachusetts

First Printing, March 1983

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

© Digital Equipment Corporation 1983.
All Rights Reserved.

Printed in U.S.A.

A postage-paid READER'S COMMENTS form is included on the last page of this document. Your comments will assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

digital™

DEC
DECmate
DECsystem–10
DECSYSTEM–20
DECUS
DECwriter
DIBOL
MASSBUS
PDP
P/OS
Professional
Rainbow
RSTS
RSX
UNIBUS
VAX
VMS
VT
Work Processor

M22600

Contents

Page

Preface xv

Part I Utility Programs

Chapter 1 Command String Interpreter (CSI)

1.1 CSI Syntax 1–1
1.2 Prompting Characters 1–3

Chapter 2 Binary File Comparison Program (BINCOM)

2.1 Calling and Terminating BINCOM 2–1
2.2 BINCOM Command String Syntax 2–2
2.3 Using Wildcards with BINCOM 2–2
2.4 Options 2–3
2.5 Output Format 2–4
2.6 Examples 2–5
2.7 Creating a SIPP Command File 2–6

Chapter 3 Backup Utility Program (BUP)

3.1 Calling and Terminating BUP 3–1
3.2 BUP Command String Syntax 3–1
3.3 Options 3–2
3.3.1 File Backup Operation 3–2
3.3.2 Volume Backup Operation (/I) 3–4
3.3.3 Directory Option (/L) 3–6
3.3.4 Restore Option (/X) 3–7
3.3.5 Initialize Option (/Z) 3–8

Chapter 4 Directory Program (DIR)

4.1 Calling and Terminating DIR 4–1
4.2 Directory Command String Syntax 4–1
4.3 Reading Directory Listings 4–2
4.4 Options 4–2

4.4.1 Alphabetical Option (/A) 4–2
4.4.2 Block Number Option (/B) 4–4
4.4.3 Columns Option (/C[:n]) 4–4
4.4.4 Date Option (/D[:date]) 4–5
4.4.5 Entire Option (/E) 4–5
4.4.6 Fast Option (/F) 4–5
4.4.7 Begin Option (/G) 4–6
4.4.8 Since Option (/J[:date]) 4–6
4.4.9 Before Option (/K[:date]) 4–6
4.4.10 Listing Option (/L) 4–7
4.4.11 Unused Areas Option (/M) 4–7
4.4.12 Summary Option (/N) 4–7
4.4.13 Octal Option (/O) 4–8
4.4.14 Exclude Option (/P) 4–8
4.4.15 Deleted Option (/Q) 4–9
4.4.16 Reverse Option (/R) 4–9
4.4.17 Sort Option (/S[:xxx]) 4–9
4.4.18 Protection Option (/T) 4–11
4.4.19 No Protection Option (/U) 4–11
4.4.20 Volume ID Option (/V[:ONL]) 4–11

Chapter 5 DUMP Program (DUMP)

5.1 Calling and Terminating DUMP 5–1
5.2 DUMP Command String Syntax 5–1
5.3 Options 5–1
5.4 Example Commands and Listings 5–3

Chapter 6 Device Utility Program (DUP)

6.1 Calling and Terminating DUP 6–1
6.2 DUP Command String Syntax 6–1
6.3 Options 6–1
6.3.1 Create Option (/C[/G:n]) 6–2
6.3.2 Image Copy Option (/I) 6–5
6.3.3 Bad Block Scan Option (/K) 6–7
6.3.4 File Option (/F) 6–8
6.3.5 Boot Option (/O) 6–9
6.3.6 Boot Foreign Volume Option (/Q) 6–10
6.3.7 Squeeze Option (/S) 6–11
6.3.8 Extend Option (/T:n) 6–12
6.3.9 Bootstrap Copy Option (/U[:xx]) 6–13
6.3.10 Volume ID Option (/V[:ONL]) 6–14
6.3.11 Wait for Volume Option (/W) 6–15
6.3.12 No Query Option (/Y) 6–15
6.3.13 Directory Initialization Option (/Z[:n]) 6–16
6.3.13.1 Changing Directory Segments (/N:n) 6–16
6.3.13.2 Changing Volume ID (/V) 6–17
6.3.13.3 Replacing Bad Blocks (/R[:RET]) 6–17
6.3.13.4 Covering Bad Blocks (/B[:RET]) 6–19
6.3.13.5 Restoring a Disk (/D) 6–19

Chapter 7 File Exchange Program (FILEX)

7.1 File Formats 7–2
7.2 Calling and Terminating FILEX 7–2
7.3 Options 7–2

7.3.1 Transferring Files Between RT–11 and DOS/BATCH or RSTS (/S) 7–4
7.3.2 Transferring Files Between RT–11 and Interchange Diskette (/U) 7–6
7.3.3 Transferring Files to RT–11 from DECsystem–10 (/T) 7–8
7.3.4 Listing Directories (/L) 7–9
7.3.5 Deleting Files from DOS/BATCH (RSTS) DECtapes and Interchange Diskettes (/D) 7–10
7.3.6 Initializing the Directories of DECtapes and Interchange Diskettes (/Z) 7–11
7.3.7 Interchange Diskette Volume ID Option (/V[:ONL]) 7–12
7.3.8 Wait Option (/W) 7–12

Chapter 8 Volume Formatting Program (FORMAT)

8.1 Calling and Terminating FORMAT 8–1
8.2 FORMAT Command String Syntax 8–2
8.3 FORMAT Confirmation Prompts 8–2
8.4 Options 8–3

8.4.1 Default Format 8–4
8.4.2 Pattern Verification Option (/P:n) 8–4
8.4.3 Single-Density Option (/S) 8–6
8.4.4 Verification Option (/V[:ONL]) 8–6
8.4.5 Wait Option (/W) 8–7
8.4.6 No Query Option (/Y) 8–7

Chapter 9 Logical Disk Subsetting Utility (LD)

9.1 Calling and Terminating LD 9–1
9.2 LD Command String Syntax 9–2
9.3 Options 9–2

9.3.1 Assign Logical Device Name Option (/A:ddd) 9–2
9.3.2 Validate Logical Disk Assignments Option (/C) 9–3
9.3.3 Define Logical Disk Option (/L:n) 9–3
9.3.4 Write-Lock Logical Disk Option (/R:n) 9–4
9.3.5 Write-Enable Logical Disk Option (/W:n) 9–4

Chapter 10 Librarian (LIBR)

10.1 Calling and Terminating LIBR 10–1
10.2 LIBR Command String Syntax 10–2

10.2.1 Creating a Library File 10–3
10.2.2 Inserting Modules into a Library 10–3
10.2.3 Merging Library Files 10–4

10.3 Option Commands and Functions for Object Libraries 10–4

10.3.1 Include All Global and Absolute Global Symbols Option (/A). 10–5
10.3.2 Command Continuation Options (/C or //). 10–6
10.3.3 Delete Option (/D) 10–6
10.3.4 Extract Option (/E). 10–7
10.3.5 Delete Global Option (/G) 10–8
10.3.6 Include Module Names Option (/N). 10–8
10.3.7 Include P-Section Names Option (/P) 10–9
10.3.8 Replace Option (/R) 10–9
10.3.9 Update Option (/U). 10–10
10.3.10 Wide Option (/W) 10–11
10.3.11 Creating Multiple Definition Libraries Option (/X) 10–11
10.3.12 Listing the Directory of a Library File 10–11
10.3.13 Combining Library Option Functions. 10–13

10.4 Option Commands and Functions for Macro Libraries 10–13

10.4.1 Command Continuation Options (/C or //). 10–14
10.4.2 Macro Option (/M[:n]) 10–14

Chapter 11 Linker (LINK)

11.1 Overview of the Linking Process 11–2

11.1.1 What the Linker Does 11–2
11.1.2 How the Linker Structures the Load Module 11–3

11.1.2.1 Absolute Section 11–3
11.1.2.2 Program Sections. 11–4

11.1.3 Global Symbols: Cummunication Links Between Modules. . . 11–7

11.2 Calling and Terminating the Linker 11–8
11.3 Link Command String Syntax 11–8
11.4 Input and Output 11–12

11.4.1 Input Object Modules 11–12
11.4.2 Input Library Modules 11–12
11.4.3 Output Load Module 11–16
11.4.4 Output Load Map 11–18

11.5 Creating an Overlay Structure 11–20

11.5.1 Low Memory Overlays 11–20
11.5.2 Extended Memory Overlays 11–28

11.5.2.1 Virtual Address Space 11–30
11.5.2.2 Physical Address Space. 11–31
11.5.2.3 Virtual and Privileged Jobs 11–32
11.5.2.4 Extended Memory Overlay Option (/V:n[:m]) 11–33

11.5.3 Combining Low Memory Overlays with Extended Memory Overlays 11–36
11.5.4 Load Map 11–38

11.6 Options 11–43

11.6.1 Alphabetical Option (/A) 11–43
11.6.2 Bottom Address Option (/B:n) 11–43
11.6.3 Continuation Option (/C or //) 11–44

11.6.4 Duplicate Global Symbol Option (/D) 11–45
11.6.5 Extend Program Section Option (/E:n) 11–47
11.6.6 Default FORTRAN Library Option (/F) 11–47
11.6.7 Directory Buffer Size Option (/G) 11–48
11.6.8 Highest Address Option (/H:n) 11–48
11.6.9 Include Option (/I) 11–49
11.6.10 Memory Size Option (/K:n) 11–49
11.6.11 LDA Format Option (/L) 11–49
11.6.12 Modify Stack Address Option (/M[:n]) 11–50
11.6.13 Cross-Reference Option (/N) 11–50
11.6.14 Low Memory Overlay Option (/O:n) 11–51
11.6.15 Library List Size Option (/P:n) 11–52
11.6.16 Absolute Base Address Option (/Q) 11–52
11.6.17 REL Format Option (/R[:n]) 11–53
11.6.18 Symbol Table Option (/S) 11–53
11.6.19 Transfer Address Option (/T[:n]) 11–54
11.6.20 Round Up Option (/U:n) 11–55
11.6.21 Extended Memory Overlay Option (/V:n[:m]) 11–55
11.6.22 Map Width Option (/W) 11–55
11.6.23 Bitmap Inhibit Option (/X) 11–56
11.6.24 Boundary Option (/Y:n) 11–56
11.6.25 Zero Option (/Z:n) 11–56

11.7 Linker Prompts 11–57

Chapter 12 MACRO–11 Program Assembly

12.1 Calling the MACRO–11 Assembler 12–1
12.2 MACRO–11 Assembler Command String Syntax 12–2
12.3 Terminating the MACRO–11 Assembler 12–3
12.4 Assigning the Temporary Work File 12–3
12.5 File Specification Options 12–4

12.5.1 Listing Control Options (/L:arg and /N:arg) 12–4
12.5.2 Function Control Options (/D:arg and /E:arg) 12–6
12.5.3 Macro Library File Designation Option (/M) 12–8
12.5.4 Cross-Reference (CREF) Table Generation Option 12–8

12.5.4.1 Obtaining a Cross-Reference Table 12–8
12.5.4.2 Handling Cross-Reference Table Files 12–10

12.6 MACRO–11 Error Codes 12–12

Chapter 13 Peripheral Interchange Program (PIP)

13.1 Calling and Terminating PIP 13–1
13.2 PIP Command String Syntax 13–1
13.3 Using Wildcards with PIP 13–2
13.4 Options 13–3

13.4.1 Operations Involving Magtape (/M:n) 13–5
13.4.2 Copy Operations 13–8

13.4.2.1 Image Mode 13–8
13.4.2.2 ASCII Mode (/A) 13–8
13.4.2.3 Binary Mode (/B) 13–9

13.4.3 Date Option (/C[:date]) 13–9

13.4.4 Delete Option (/D) 13–9
13.4.5 Wait Option (/E) 13–10

13.4.5.1 Single-Drive Operation 13–11
13.4.5.2 Double-Drive Operation 13–11

13.4.6 Protection Option (/F) 13–12
13.4.7 Ignore Errors Option (/G) 13–13
13.4.8 Verify Option (/H) 13–13
13.4.9 Since Option (/I[:date]) 13–14
13.4.10 Before Option (/J[:date]) 13–14
13.4.11 Copies Option (/K:n) 13–14
13.4.12 No Replace Option (/N) 13–14
13.4.13 Predelete Option (/O) 13–14
13.4.14 Exclude Option (/P) 13–15
13.4.15 Query Option (/Q) 13–15
13.4.16 Rename Operation (/R) 13–15
13.4.17 Single-Block Transfer Option (/S) 13–16
13.4.18 Set Date Option (/T[:date]) 13–16
13.4.19 Concatenate Option (/U) 13–16
13.4.20 Multivolume Option (/V) 13–17
13.4.21 Logging Option (/W) 13–17
13.4.22 Information Option (/X) 13–18
13.4.23 System Files Option (/Y) 13–18
13.4.24 No Protection Option (/Z) 13–18

Chapter 14 Resource Utility Program (RESORC)

14.1 Calling and Terminating RESORC 14–1
14.2 Options 14–2

14.2.1 All Option (/A) 14–2
14.2.2 Software Configuration Option (/C) 14–3
14.2.3 Device Handler Status Option ([dd:]/D) 14–3
14.2.4 Hardware Configuration Option (/H) 14–5
14.2.5 Loaded Jobs Option (/J) 14–5
14.2.6 Device Assignments Option (/L) 14–6
14.2.7 Current Monitor Option (/M) 14–7
14.2.8 Special Features Option (/O) 14–7
14.2.9 Disk Subsetting Option (/S) 14–8
14.2.10 Terminal Status Option (/T) 14–8
14.2.11 Physical Memory Layout Option (/X) 14–9
14.2.12 Summary Option (/Z) 14–10

Chapter 15 Source Comparison (SRCCOM)

15.1 Calling and Terminating SRCCOM 15–1
15.2 SRCCOM Command String Syntax 15–1
15.3 Using Wildcards with SRCCOM 15–2
15.4 Options 15–3
15.5 Differences Listing Format 15–3

15.5.1 Sample Text 15–3
15.5.2 Sample Differences Listing 15–4
15.5.3 Changebar Option (/D[/V:i:d]) 15–7

15.6 Creating a SLP Command File 15–8

Part II System Jobs

Chapter 16 Error Logging Subsystem

16.1 Uses 16–1
16.2 Error Logging Subsystem 16–2
16.3 Calling and Using the Error Logger with the SJ Monitor 16–5
16.4 Calling and Using the Error Logger with the FB or XM Monitor 16–6
16.4.1 Using ELINIT 16–6
16.5 Using ERROUT 16–7
16.6 Report Analysis 16–8
16.6.1 Storage Device Error Report 16–9
16.6.2 Memory Error Report 16–10
16.6.3 Summary Error Report 16–12

Chapter 17 Queue Package

17.1 Calling and Using the Queue Package 17–1
17.1.1 Running QUEUE 17–1
17.1.2 Running QUEMAN 17–2
17.2 QUEMAN Options 17–2
17.2.1 Terminating QUEUE (/A) 17–4
17.2.2 Date Option (/C[:date]) 17–4
17.2.3 Deleting Input Files After Printing (/D) 17–5
17.2.4 Printing Banner Pages (/H:n) 17–5
17.2.5 Since Option (/I[:date]) 17–6
17.2.6 Before Option (/J[:date]) 17–6
17.2.7 Printing Multiple Copies (/K:n) 17–6
17.2.8 Listing the Contents of the Queue (/L) 17–6
17.2.9 Removing a Job from the Queue (/M) 17–7
17.2.10 No Banner Pages Option (/N) 17–7
17.2.11 Setting Queue Package Defaults (/P) 17–7
17.2.12 Query Option (/Q) 17–8
17.2.13 Suspending Output (/S) 17–8
17.2.14 Resuming/Restarting Output (/R) 17–9
17.2.15 Log Option (/W) 17–9
17.2.16 Information Option (/X) 17–9
17.2.17 Continuing a Command String (//) 17–10

Part III Debugging and Altering Programs

Chapter 18 On-Line Debugging Technique (ODT)

18.1 Calling and Using ODT 18–1
18.2 Relocation 18–5
18.3 Commands and Functions 18–6
18.3.1 Printout Formats 18–6
18.3.2 Opening, Changing, and Closing Locations 18–7

18.3.2.1 Slash (/) 18–7
18.3.2.2 Backslash (\) 18–8
18.3.2.3 LINE FEED Key (LF) 18–8
18.3.2.4 Circumflex or Up-Arrow (^) 18–8
18.3.2.5 Underline or Back-Arrow (_ or ←) 18–8
18.3.2.6 Open the Addressed Location (@) 18–9
18.3.2.7 Relative Branch Offset (>) 18–9
18.3.2.8 Return to Previous Sequence (<) 18–9

18.3.3 Accessing General Registers 0–7 18–9
18.3.4 Accessing Internal Registers 18–10
18.3.5 Radix–50 Mode (X) 18–11
18.3.6 Breakpoints 18–12
18.3.7 Running the Program (r;G and r;P) 18–13
18.3.8 Single-Instruction Mode 18–14
18.3.9 Searches 18–15

18.3.9.1 Word Search (r;W) 18–15
18.3.9.2 Effective Address Search (r;E) 18–16

18.3.10 Constant Register (r;C) 18–16
18.3.11 Memory Block Initialization (;F and ;I) 18–16
18.3.12 Calculating Offsets (r;O) 18–17
18.3.13 Relocation Register Commands 18–18
18.3.14 The Relocation Calculators, n! and nR 18–19
18.3.15 ODT Priority Level ($P) 18–19
18.3.16 ASCII Input and Output (r;nA) 18–20

18.4 Programming Considerations 18–21

18.4.1 Using ODT with Foreground/Background Jobs 18–21
18.4.2 Functional Organization 18–21
18.4.3 Breakpoints 18–22
18.4.4 Searches 18–25
18.4.5 Terminal Interrupt 18–25

18.5 Error Detection 18–26

Chapter 19 Object Module Patch Program (PAT)

19.1 Calling and Using PAT 19–1
19.2 PAT Command String Syntax 19–2
19.3 How PAT Effects Updates 19–4

19.3.1 Input File 19–4
19.3.2 Correction File 19–4

19.4 Updating Object Modules 19–5

19.4.1 Overlaying Lines in a Module 19–5
19.4.2 Adding a Subroutine to a Module 19–6

19.5 Determining and Validating the Contents of a File 19–8

Chapter 20 Save Image Patch Program (SIPP)

20.1 Calling and Using SIPP 20–1
20.2 SIPP Options 20–2
20.3 SIPP Dialog 20–2

20.4 SIPP Commands 20–4

20.4.1 Opening and Modifying Locations Within a File 20–6
20.4.2 Backing Up Through Files 20–6
20.4.3 Advancing in Bytes 20–6
20.4.4 Entering Octal Values (;O) 20–7
20.4.5 Displaying and Entering ASCII Values 20–7
20.4.6 Displaying and Entering Radix–50 Values 20–8
20.4.7 Searching Through Files (;S) 20–9
20.4.8 Verifying (;V) 20–10
20.4.9 Backing Up to a Previous Prompt 20–11
20.4.10 Completing Code Modifications 20–12
20.4.11 Extending Files and Overlay Segments 20–12

20.4.11.1 Nonoverlaid Program 20–13
20.4.11.2 Overlaid Program, Low Memory Overlays Only . . . 20–13
20.4.11.3 Overlaid Program, Extended Memory Overlays Only 20–13
20.4.11.4 Overlaid Program, Both Low Memory and Extended Memory Overlays 20–14

20.5 SIPP Checksum 20–15
20.6 Running SIPP from an Indirect File 20–16
20.7 Running SIPP from a BATCH Stream 20–17

Chapter 21 Source Language Patch Program (SLP)

21.1 Calling and Terminating SLP 21–1
21.2 SLP Command String Syntax 21–1
21.3 Options 21–2
21.4 Example 21–2
21.5 Creating and Maintaining a Command File 21–4

21.5.1 Update Line Format 21–4
21.5.2 Creating a Numbered Listing 21–6
21.5.3 Adding Lines to a File 21–7
21.5.4 Deleting Lines in a File 21–9
21.5.5 Replacing Lines in a File 21–10
21.5.6 Determining and Validating the Contents of a File 21–11

Appendix A BATCH

A.1 Hardware and Software Requirements A–1
A.2 Control Statement Format A–2

A.2.1 Command Fields A–2

A.2.1.1 Command Names A–3
A.2.1.2 Command Field Options A–3

A.2.2 Specification Fields A–3

A.2.2.1 Physical Device Names A–6
A.2.2.2 File Specifications A–6
A.2.2.3 Wildcard Construction A–6
A.2.2.4 Specification Field Options A–7

A.2.3 Comment Fields A–7
A.2.4 BATCH Character Set A–8

A.2.5 Temporary Files A–10

A.3 General Rules and Conventions A–11
A.4 Commands. A–11

A.4.1 $BASIC A–13
A.4.2 $CALL A–14
A.4.3 $CHAIN A–14
A.4.4 $COPY A–15
A.4.5 $CREATE A–16
A.4.6 $DATA A–17

A.4.6.1 Using $DATA with FORTRAN Programs. A–18

A.4.7 $DELETE A–18
A.4.8 $DIRECTORY A–19
A.4.9 $DISMOUNT A–19
A.4.10 $EOD A–20
A.4.11 $EOJ A–20
A.4.12 $FORTRAN A–21
A.4.13 $JOB A–23
A.4.14 $LIBRARY A–24
A.4.15 $LINK. A–25
A.4.16 $MACRO A–26
A.4.17 $MESSAGE A–29
A.4.18 $MOUNT A–30
A.4.19 $PRINT A–31
A.4.20 $RT11 . . . . . A–32
A.4.21 $RUN A–32
A.4.22 $SEQUENCE A–33
A.4.23 Sample BATCH Stream A–33

A.5 RT–11 Mode A–35

A.5.1 Communicating with RT–11 . . . . . A–36
A.5.2 Creating RT–11 Mode BATCH Programs. A–37

A.5.2.1 Labels A–37
A.5.2.2 Variables A–37
A.5.2.3 Terminal I/O Control. A–40
A.5.2.4 Other Control Characters. A–40
A.5.2.5 Comments A–41

A.5.3 RT–11 Mode Examples. A–41

A.6 Creating BATCH Programs on Punched Cards A–42
A.7 Operating Procedures A–43

A.7.1 Loading BATCH. A–43
A.7.2 Running BATCH A–45
A.7.3 Communicating with BATCH Jobs. A–48
A.7.4 Terminating BATCH. A–51

A.8 Differences Between RT–11 BATCH and RSX–11D BATCH A–51

Appendix B System Utility Program Options and Monitor Command Equivalents

Index

Figures

11–1 Library Searches 11–14
11–2 Sample Load Map. 11–18
11–3 Sample Overlay Structure for a FORTRAN Program 11–21
11–4 Overlay Scheme 11–22
11–5 Run-Time Overlay Handler — Low Memory 11–23
11–6 Sample Subroutine Calls and Return Paths 11–25
11–7 Memory Diagram Showing BASIC Link with Overlay Regions 11–29
11–8 Program Virtual Address Space 11–30
11–9 Physical Address Space for Program with Low Memory Overlays. 11–31
11–10 Virtual and Physical Address Space. 11–32
11–11 Virtual and Physical Address Space. 11–34
11–12 Extended Memory Partitions that Contain Sharing Segments 11–35
11–13 Memory Diagram Showing Low Memory and Extended Memory Overlays. 11–37
11–14 Load Map for Program with Unmapped and Virtual Overlays 11–38
11–15 Extended Memory Overlay Handler. 11–41
11–16 Global Data Section with CON Attribute 11–46
11–17 Global Data Section with OVR Attribute 11–47
12–1 Sample Assembly Listing 12–5
12–2 Cross-Reference Table 12–11
16–1 Error Logging Subsystem — FB and XM. 16–4
16–2 Error Logging Subsystem — SJ 16–4
16–3 Sample Storage Device Error Report 16–9
16–4 Sample Memory Parity Error Report 16–10
16–5 Sample Cache Memory Error Report 16–11
16–6 Sample Summary Error Report for Device Statistics 16–12
16–7 Sample Summary Error Report for Memory Statistics 16–13
16–8 Sample Report File Environment and Error Count Report 16–13
18–1 Linking ODT with a Program. 18–2
19–1 Updating a Module Using PAT 19–2
19–2 Processing Steps Required to Update a Module Using PAT 19–3
A–1 EOF Card A–43

Tables

1–1 Prompting Characters 1–3
2–1 BINCOM Options. 2–3
3–1 BUP Options 3–2
4–1 DIR Options 4–3
4–2 Sort Codes 4–10

5–1 DUMP Options 5–2
6–1 DUP Option Combinations 6–2
6–2 DUP Options 6–3
6–3 Default Directory Sizes 6–17
7–1 Supported FILEX Devices. 7–1
7–2 FILEX Options 7–3
8–1 FORMAT Options 8–3
8–2 Verification Bit Patterns 8–5
9–1 LD Options 9–2
10–1 LIBR Object Options 10–5
10–2 LIBR Macro Options 10–14
11–1 P-Sect Attributes 11–5
11–2 Section Attributes 11–6
11–3 P-Sect Order 11–6
11–4 Global Reference Resolution 11–7
11–5 Linker Defaults. 11–9
11–6 Linker Options 11–10
11–7 Absolute Block Parameters 11–17
11–8 Line-by-Line Sample Load Map Description 11–19
11–9 Line-by-Line Sample Load Map Description 11–39
11–10 Linker Prompting Sequence. 11–57
12–1 Default File Specification Values 12–3
12–2 File Specification Options. 12–4
12–3 Arguments for /L and /N Listing Control Options 12–6
12–4 Arguments for /E and /D Function Control Options 12–7
12–5 /C Option Arguments 12–9
12–6 MACRO–11 Error Codes 12–12
13–1 PIP Options 13–4
14–1 RESORC Options 14–2
15–1 SRCCOM Options. 15–4
16–1 Line-by-Line Analysis of the Sample Storage Device Error Report 16–10
16–2 Line-by-Line Analysis of the Sample Memory Error Report 16–11
17–1 QUEMAN Options 17–3
18–1 Forms of Relocatable Expressions (r) 18–6
18–2 Internal Registers 18–10
18–3 Radix–50 Terminators 18–11
18–4 Single-Instruction Mode Commands. 18–14
18–5 ASCII Terminators 18–20
20–1 SIPP Options 20–3
20–2 SIPP Commands 20–4
20–3 Overlaid Program Segment Limits 20–14
20–4 Overlaid Program Segment Limits 20–14
20–5 Overlaid Program Segment Limits 20–15
21–1 SLP Options 21–3
21–2 SLP Command File Operators. 21–5
A–1 Command Field Options A–4
A–2 BATCH File Types A–7
A–3 Specification Field Options A–8
A–4 Character Explanation A–9
A–5 BATCH Commands. A–12
A–6 Operator Directives to BATCH Run-Time Handler A–49
A–7 Differences Between RT–11 and RSX–11D BATCH A–51
B–1 System Program/Monitor Command Equivalents B–1

Preface

This manual describes how to use the RT–11 system utilities. You can use the RT–11 system utilities instead of the keyboard monitor commands described in the *RT–11 System User's Guide* to perform program development, program execution, and file maintenance.

The manual is written for you if you are already familiar with computer software fundamentals and have some experience using RT–11 and RT–11 keyboard monitor commands. If you have no RT–11 experience, you should first read the *Introduction to RT–11* and the *RT–11 System User's Guide* before consulting this manual. If you have experience with an earlier release of RT–11 (this is Version 5), you should read the *RT–11 System Release Notes* to learn how RT–11 Version 5 differs from earlier versions. If you are interested in more sophisticated programming techniques or in system programming, you should read this manual first and then proceed to the *RT–11 Programmer's Reference Manual* and the *RT–11 Software Support Manual.*

The next section, Chapter Summary, briefly describes the chapters in this manual and suggests a reading path to help you use the manual efficiently.

Chapter Summary

Part I, Utility Programs, Chapters 1–15, describes the Command String Interpreter (CSI) and many utility programs provided with the RT–11 system. These programs include:

BINCOM	Binary file comparison program
BUP	Backup utility program
DIR	Directory program
DUMP	Dump program
DUP	Device utility program
FILEX	File exchange program
FORMAT	Volume formatting program
LD	Logical disk subsetting program
LIBR	Librarian program
LINK	Linker program
MACRO	MACRO–11 assembler
PIP	Peripheral interchange program
RESORC	Resource program
SRCCOM	Source comparison program

Part II, System Jobs, Chapters 16–17, describes two utilities that run under the foreground/background (FB) or extended memory (XM) monitor as foreground or system jobs: the Error Logger and the Queue Package. To run them as system jobs, you must enable system job support through the system generation process. The Error Logger also runs under the single-job (SJ) monitor.

Part III, Debugging and Altering Programs, Chapters 18–21, describes the four utility programs that permit you to examine and change assembled programs and source files. These utilities are:

ODT	On-line debugging technique
PAT	Object module patch program
SIPP	Save image patch program
SLP	Source language patch program

Appendix A describes BATCH processing. Appendix B contains a summary of the system utility programs and their keyboard monitor command equivalents.

Documentation Conventions

A description of the symbolic conventions used throughout this manual follows. Familiarize yourself with these conventions before you continue reading.

1. Wherever possible, examples appear as if they are computer output. What you should type appears in red.

2. This manual uses the symbol (RET) to represent a carriage return, (LF) to represent a line feed, (SP) for a space, and (TAB) to represent a tab. Unless the manual indicates otherwise, terminate all commands or command strings with a carriage return.

3. Terminal and console terminal are general terms used throughout all RT–11 documentation to represent any terminal device, including DECwriters and video terminals.

4. To produce certain characters in system commands, you must type a letter key while pressing the control (CTRL) key. For example, while holding down the CTRL key, type C to produce the CTRL/C character. Key combinations of this type are documented as (CTRL/ C), (CTRL/ O), and so on.

5. In discussions of command syntax, uppercase letters represent the command name, which you must type. Lowercase letters represent a variable, for which you must supply a value.

Square brackets ([ ]) enclose options; you may include the item in brackets, or you may omit it, as you choose.

The ellipsis symbol (...) represents repetition. You can repeat the item that precedes the ellipsis.

This is a typical illustration of command syntax:

.R PIP
*out-filespec[/option...] = in-filespec[/option...]

This example shows that you must run the utility program PIP as shown, and enter file specifications and options of your choice (none are required) on the line that follows. The first file specification represents the output file, and the second represents the input file. Here is a typical command string:

```
.R PIP
*DL1:MYFIL.BAK=DL0:MYFIL.MAC/A/J
```

Part I
Utility Programs

Part I of this manual presents, in alphabetical order, most of the utility programs available with RT–11. You can take advantage of nearly all of the capabilities of RT–11 by using the keyboard commands (described in Chapter 4 of the *RT–11 System User's Guide*), but it is the utility programs that actually perform many of the system's functions. For example, when you issue the CREATE command, the utility program DUP performs the create operation.

This part of the manual explains how to carry out utility operations, those not performed directly by the monitor, by running a specific utility program instead of using the keyboard monitor commands. It is not necessary to have an understanding of the material contained in Part I of this manual in order to use the RT–11 system. However, the information in this part may be of interest to you if you have experience with a previous version of RT–11, or if you are a systems programmer and need to perform certain functions with the utility programs that are not available with the keyboard monitor commands.

Note that the syntax required by the Command String Interpreter for input and output specifications is different from the syntax you use to issue a keyboard monitor command. Chapter 1, Command String Interpreter, describes the general syntax of the command string that the system utility programs accept, and explains certain conventions and restrictions. Read this chapter carefully before you use any of the system utility programs directly, and bear in mind that there are many differences between issuing a keyboard command and running a utility program. Chapters 2 through 15 describe the system utility programs themselves.

Chapter 1
Command String Interpreter (CSI)

The Command String Interpreter (CSI) is the part of RT–11 that accepts a line of ASCII input, usually from the console terminal, and interprets it as a string of input specifications, output specifications, and options for use by a utility program.

To call a utility program,respond to the dot (.) printed by the keyboard monitor by typing R followed by a program name and a carriage return. This example shows how to call the directory program (DIR):

```
.R DIR (RET)
*
```

The CSI prints an asterisk (*) at the left margin of the terminal, indicating that it is ready to accept a list of specifications and options. The following section describes the syntax of the specifications and options you can enter.

You can use the single-line editor, described in Section 4.3 of the *RT–11 System User's Guide*, to edit CSI command strings and terminal input as well as keyboard monitor commands.

1.1 CSI Syntax

Once you have started a system program, you must enter the appropriate information before any operation can be performed. You type a specification string with the following general syntax in response to the prompting asterisk:

output-filespecs/options = input-filespecs/options

A few system programs — BINCOM, for example — require you to enter this information differently. Complete instructions are provided in the appropriate chapters.

In all cases, the syntax for output-filespecs is:

dev:filnam.typ[n],...dev:filnam.typ[n]

The syntax for input-filespecs is:

dev:filnam.typ,...dev:filnam.typ

The syntax for /option is:

/o[:oval]

or

/o[:dval].

where:

dev: represents either a logical device name or one of the physical device names from Table 3–1 in the *RT–11 System User's Guide.*

If you do not supply a device name, the system uses device DK:. DK:, or whatever device you specify for the first file in a list of input or output files, applies to all the files in that input or output list until you supply a different device name. For example:

```
*DY1:FIRST.OBJ,LP:=TASK.1,DL1:TASK.2,TASK.3
```

This command is interpreted as follows:

```
*DY1:FIRST.OBJ,LP:=DK:TASK.1,DL1:TASK.2,DL1:TASK.3
```

File FIRST.OBJ is stored on device DY1:. File TASK.1 is stored on default device DK:. Files TASK.2 and TASK.3 are stored on device DL1:. Notice that file TASK.1 is on device DK:. It is the first file in the input file list and the system uses the default device DK:. Device DY1: applies only to the file on the output side of the command.

filnam.typ is the name of a file (consisting of one to six alphanumeric characters followed optionally by a period and a zero- to three-character file type). No spaces or tabs are allowed in the file name or file type. As many as three output and six input files are allowed. If you omit the dot and the file type, the system may apply a default file type that the program specifies.

[n] is an optional declaration of the number of blocks you need for an output file; n is a decimal number (up to 65,535) enclosed in square brackets immediately following the output filnam.typ to which it applies.

/o[:oval] or /o[:dval]. is one or more options whose functions vary according to the program you are using (refer to the option table in the appropriate chapter). The variable oval is either an octal number or one to three alphanumeric characters (the first of which must be alphabetic) that the program converts to Radix–50 characters. The variable dval. is a decimal number followed by a decimal point. You can use a minus sign (–) to denote negative octal or decimal numbers.

This manual uses the /o:oval construction throughout, except for the keyboard monitor commands, where all values are interpreted as decimal (unless indicated otherwise) and the decimal point after a value is not necessary. However, the /o:dval. format is always valid. Generally, these options and their associated values, if any, should follow the device and file name to which they apply.

If the same option is to be repeated several times with different values (for example, /L:MEB/L:TTM/L:CND) you can abbreviate the line as /L:MEB:TTM:CND. You can mix octal, Radix–50, and decimal values.

= is a delimiter that separates the output and input fields. You can use the < sign in place of the = sign. You can omit this separator entirely if there are no output files.

NOTE

Except where noted, all numeric values you supply to the CSI must be in octal.

Concise Command Language (CCL) also uses CSI syntax. See Section 4.6 of the *RT–11 System User's Guide* for information on using CCL.

1.2 Prompting Characters

Table 1–1 summarizes the characters RT–11 prints either to indicate that the system is waiting for your response or to specify which job (foreground, system, or background) is producing output.

Table 1–1: Prompting Characters

Character	Explanation
. (dot)	The keyboard monitor is waiting for a command.
^	When the console terminal is being used as an input file, the circumflex prompts you to enter information from the keyboard. Typing a CTRL/Z marks the end-of-file. See Section 3.6 of the *RT–11 System User's Guide* for details on special function keys.
>	If a foreground or system job is active, the > character identifies which job, foreground, system, or background, is producing the output that currently appears on the console terminal. Each time output from the background job is to appear, B> prints first, followed by the output. If the foreground job is to print output, F> prints first. If a system job is to print output, jobname> appears first, where jobname represents the name of the system job.
*	The current system utility program is waiting for a line of specifications and options.

Chapter 2
Binary File Comparison Program (BINCOM)

The RT–11 binary comparison program (BINCOM) compares two volumes or binary files and lists the differences between them. BINCOM can either print the results at the terminal or line printer, or store them in a file. BINCOM is particularly useful when you need to compare two executable programs, because it provides a quick way of telling whether two data files are identical. Another use of BINCOM is to verify whether two versions of a program produce identical output files when given identical input files.

BINCOM examines the two input files word by word (or byte by byte), looking for differences. When BINCOM finds a mismatch, it prints the block number and offset within the block at which the difference occurs, the octal values from each input file, and the logical exclusive OR of the two values. This last number helps you find the bits that are different in the two values.

You can also use BINCOM to create an indirect command file that invokes the save image patch program (SIPP, described in Chapter 20) to patch one version of a file so that it matches another version. Section 2.6 describes the procedure you can use to create an indirect command file for SIPP.

2.1 Calling and Terminating BINCOM

To call BINCOM from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
. R BINCOM (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by entering only a carriage return, BINCOM prints its current version number. You can type a CTRL/C to halt BINCOM and return control to the monitor when BINCOM is waiting for input from the console terminal. You must type two CTRL/Cs to abort BINCOM at any other time. To restart BINCOM, type R BINCOM or REENTER and a carriage return in response to the monitor's dot.

2.2 BINCOM Command String Syntax

BINCOM accepts command strings with the following syntax:

[output-spec[/option]][,patch-spec[/option]] = old-filespec, new-filespec[/option...]

where:

output-spec	represents the file or volume to which you want the differences between the two files or volumes you are comparing sent. If omitted, the default is TT:.
patch-spec	represents the file that you can run as an indirect command file; it will contain the commands necessary to patch old-filespec so it matches new-filespec.
old-filespec	represents the first file to be compared.
new-filespec	represents the second file to be compared.
option	is one or more of the options listed in Table 2–1.

The console terminal is the default output device. There is no default file type for input files; you must always specify the file type. BINCOM assigns .DIF as the default file type for the difference output file, and .COM as the default file type for the SIPP indirect command file.

2.3 Using Wildcards with BINCOM

You can use wildcards to perform multiple binary file comparisons by typing only one command line. However, you can use wildcards only to compare files; you cannot use wildcards when creating a SIPP indirect command file.

You can use wildcards in either input file specification (old-filespec or new-filespec). A different type of comparison is performed depending on whether you use wildcards in only one or in both of the input file specifications.

If you use wildcards in only one of the input file specifications, BINCOM compares the file you specify without any wildcards to all variations of the file specification that contains wildcards. The wildcards represent the part of the file specification to be varied. You can use this method to compare one particular file to several other files. For example, when the following command line is executed, BINCOM compares the file TEST1.SAV on device DY0: to all files on device DY1: with the filename TEST2:

```
* TEST=DY0:TEST1.SAV,DY1:TEST2.*
```

You can send the results of all the comparisons to a file on a volume rather than to the console by specifying an output file. In the last example, all differences from the comparisons are sent to the file TEST.DIF on device DK:.

If you use wildcards in both input file specifications, the wildcards represent a part of the file specifications that you want to be the same in both files being compared. You can use this method to compare several pairs of files; each input file specification is compared to only one other input file specification. For example, when the following command line is executed, BINCOM compares pairs of files; the first input file in each pair has the file name PROG1, and the second has the file name PROG2. The file type of both files in each pair must match.

```
*  DY0:PROG1.*,DY1:PROG2.*
```

BINCOM searches for the first file on DY0: with the file name PROG1, and takes note of its file type. Then, BINCOM searches DY1: for a file with the file name PROG2 and the same file type as PROG1. If a match is found, BINCOM compares the two files and lists the differences on the console (or sends the differences to an output file if one is specified). BINCOM then searches DY0: for more files with the file name PROG1 and DY1: for PROG2 files with matching file types.

2.4 Options

Table 2–1 summarizes the options that you can use with BINCOM. Except for the /O option, you can place these options anywhere in the command string, but it is conventional to place them at the end of the command string.

Table 2–1: BINCOM Options

Option	Function
/B	Compares the input files byte by byte. If you do not specify this option, BINCOM compares the files word by word.
/D	Compares two entire volumes starting with block 0. If one volume is longer, BINCOM prints a message and compares the volumes up to the point where the shorter volume ends and the longer one continues. Invalid when creating a SIPP command file.
/E:n	Ends comparison at block n, where n is an octal value. If you do not include this option, BINCOM ends the comparison when it reaches end-of-file on one of the input files, or end-of-device on one of the input devices.
/H	Types on the console terminal the list of available options.
/O	Creates an output file or patch file, even if there are no differences between the two input files. If you enter this option after the differences output file, BINCOM creates the differences output file whether or not there are differences between the two input files. If you enter this option after the SIPP indirect command file, BINCOM creates a SIPP indirect command file whether or not differences exist. You can enter this option at the end of the command line if you want both output files. This option is useful in BATCH streams to prevent later job steps from failing because BINCOM did not create the expected control file.

(Continued on next page)

Table 2–1: BINCOM Options (Cont.)

Option	Function
/Q	Suppresses the printing of the differences and prints only the message ?BINCOM-W-Files are different or ?BINCOM-W-Devices are different if applicable (or ?BINCOM-I-No differences found). This option is useful in BATCH control files when you want to test for differences and perhaps abort execution, but do not want the log file filled with output.
/S:n	Starts the comparison at block n, where n is an octal value.

2.5 Output Format

This section describes the BINCOM output file format and explains how to interpret it.

If you include an output file specification in the command line, BINCOM creates a file that contains the differences between the two input files or devices. If you do not specify an output file, BINCOM prints the differences only on the terminal. If you include the /Q option, BINCOM does not print the differences and does not create an output file.

The first line of the difference listing is a header line that identifies the files or devices you are comparing. Next, BINCOM prints a blank line and then lists the differences between the two files or devices. Each difference line has the following format:

bbbbbb ooo/ ffffff ssssss xxxxxx

where:

bbbbbb	is the octal number of the block that contains the difference
ooo	is the octal offset within the block
ffffff	is the value in the first file or device
ssssss	is the value in the second file or device
xxxxxx	is the logical exclusive OR of the two values

If there are several differences in a block, BINCOM prints the block number only once for that block. Thus, each time you see a block number appear, it indicates that the differences being printed are in a new block.

If you specify the /B option to compare byte by byte, BINCOM prints ffffff, ssssss, and xxxxxx as three-digit, octal byte values.

When BINCOM reaches the end of one of the input files or devices, it checks its position in the other. If the files or devices have different lengths, BINCOM prints the message:

```
?BINCOM-W-File is longer DEV:FILNAM.TYP
```

or

```
?BINCOM-W-Device is longer DEV:
```

BINCOM prints the following message on the terminal if it encountered any differences:

```
?BINCOM-W-Files are different
```

or

```
?BINCOM-W-Devices are different
```

If the two files or devices are identical up to that point, BINCOM prints this message:

```
?BINCOM-I-No differences found
```

If you include a SIPP indirect command file specification in the command line, BINCOM creates a file that is a valid command file for the save image patch program (see Chapter 20). This command file contains commands that instruct SIPP to patch the first input file so that it matches the second input file. If you want BINCOM to create only the patch file, enter a comma before the patch file specification in the command line, in place of the output file specification.

2.6 Examples

The first example compares files TEST1.TST and TEST3.TST, both on device DK:. Notice that there are no output files and no options in the command line.

```
. R BINCOM
* TEST1.TST,TEST3.TST
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

Notice the fourth line in the above example. The third number, 051511, represents the contents of location 2, block 0, in file TEST1.TST. The fourth number, 051502, represents the contents of the same location in file TEST3.TST. The last number is the logical exclusive OR of the two values.

The next example specifies the output file FOO1 as the file in which to store the differences between TEST1.TST and TEST3.TST.

```
. R BINCOM
* FOO1=TEST1.TST,TEST3.TST
?BINCOM-W-Files are different
```

The contents of file FOO1 from the last example follow. Note that FOO1 has the default file type .DIF.

```
.TYPE FOO1.DIF
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
```

2.7 Creating a SIPP Command File

You can use BINCOM to create an indirect command file that invokes the save image patch program (SIPP, described in Chapter 20) to patch one version of a file you are comparing to match the other version. As noted earlier in this chapter, you specify this indirect file as the second output file in the CSI command string. If you wish to create only the indirect file as output, place a comma before the output file specification in the command line, in place of the first output file specification.

The example that follows specifies FOO2 as the patch output file, which will contain the commands necessary to patch file TEST1.TST so it matches TEST3.TST. Notice the comma that appears before the patch file specification. This indicates that a difference output file is not requested, resulting in the printing of all the differences at the terminal when the command is executed.

```
.R BINCOM
* FOO2=TEST1.TST,TEST3.TST
BINCOM comparing/DK:TEST1.TST -- DK:TEST3.TST

000000  002/   051511  051502  000013
?BINCOM-W-Files are different
```

The contents of file FOO2 follow. Note that BINCOM assigns to this file the .COM file type.

```
.TYPE FOO2.COM
R SIPP
DK:TEST1.TST/A
000000
000000002
051502
^Y
^C
```

The file FOO2 from the previous example can be run as an indirect command file to make TEST1.TST match TEST3.TST. This can be done with the following command, when typed in response to the keyboard monitor dot:

```
@FOO2.COM
```

Chapter 3
Backup Utility Program (BUP)

The backup utility program (BUP) is a specialized file transfer program for storing large files or volumes. BUP allows you to copy a file or volume to several volumes that are smaller than the input file or volume. BUP also performs the reverse operation of restoring the fragmented file or volume to its original form on a single large volume.

Since you cannot use the file or volume while it is fragmented on several smaller volumes, BUP is most useful as a means of backing up information that you want to store.

3.1 Calling and Terminating BUP

To call BUP from the system device, respond to the keyboard monitor prompt (.) by typing:

```
. R BUP (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return at this point, BUP prints its current version number and prompts you again for a command string. You can type CTRL/C to terminate BUP and return to the monitor when BUP is waiting for input from the console terminal. You must type two CTRL/Cs to terminate BUP at any other time.

3.2 BUP Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line BUP accepts, but you can type only one input specification and only one output specification. You must specify the input file name and type. DK: is the default device for both input and output. Wildcards are ignored when initializing or obtaining the directory of a backup volume. Wildcards are not allowed in either the input or output file specification for backup and restore operations.

You can use random-access volumes as either input or output volumes for both backup and restore operations. Magtapes, however, can be used only as output volumes for a backup operation, and only as input volumes for a restore operation. If you use TSV05 magtapes as backup volumes, you must set the extended features switch (switch S0 on switch pack E58) if you want the tape to stream at 100 in/s. See Appendix A of either the *TSV05 Installation Guide* or the *TSV05 User's Guide* for more information on setting the Extended Features Switch.

Output volumes for backup operations, except magtape, must be initialized by BUP. See Section 3.3.5 for information on initializing backup volumes.

3.3 Options

BUP options, summarized in Table 3–1, permit you to perform various operations with BUP. If you specify none of these options, BUP assumes that you want to back up a file to smaller volumes.

Table 3-1: BUP Options

Option	Section	Function
/I	3.3.2	Backs up an entire volume to smaller volumes in image mode. Also used with /X during restore operations.
/L	3.3.3	Prints a directory listing of a backup volume. Cannot be used with any other option.
/X	3.3.4	Restores a file that has been backed up using BUP. Use with /I to restore a volume.
/Y	3.3.5	Used with /Z to suppress the BUP confirmation message printed during initialization.
/Z	3.3.5	Initializes a volume specifically for use as an output volume in a backup operation. Cannot be used with any other option except /Y.

3.3.1 File Backup Operation

When you specify no options in the BUP command line, BUP assumes that you want to back up a file. BUP performs all copy operations in image mode.

You must first initialize the output volumes, unless the output volumes are magtape, by using the BUP /Z option, so BUP can recognize the volumes as backup volumes and to ensure that the volumes contain no bad blocks. (The /Z option is invalid with magtape; BUP initializes magtapes during the backup operation.) See Section 3.3.5 for details on initializing volumes for BUP operations.

Use the following syntax to perform a file backup operation.

output-spec = input-spec

where:

output-spec — represents the device in which you will mount the output volumes for the backup operation, and the file to which you are copying the input file. If you specify no output file name, BUP uses the input file name. The default output file type is .BUP. Wildcards are invalid in the output specification.

input-spec — represents the device and file specification of the file you want to back up. Wildcards are invalid in the input specification.

BUP copies the input file to the first output volume until the output volume is full. If the entire input file fits on the first output volume, BUP prints an error message unless the output volume is magtape.

```
?BUP-F-Enough space on one volume -- use PIP
```

When the first of the output volumes is full, BUP prompts you to mount the next output volume. BUP also tells you which volume BUP is creating so you can properly label each volume.

```
Mount output volume in <dev>; Continue?  Y
?BUP-I-Creating volume n
```

BUP repeats this process until the entire input file has been copied.

The output file's creation date recorded in the directory is the system date during the backup operation. If no system date was set, no creation date is entered in the directory.

If BUP detects an output volume that has not been initialized as a backup volume (see Section 3.3.5), BUP prints a message and allows you to either initialize the volume as a backup volume or replace that volume with an already initialized backup volume.

```
?BUP-W-Not a backup volume DEV:
<dev:>/BUP Initialize; Are you sure?  Y
```

If you are using magtape for the backup volumes, BUP always attempts to initalize the output volume, and asks you to confirm the initialization.

If BUP detects a volume that is not a valid RT–11 volume, or a volume that already contains files, BUP prints the appropriate message and allows you to either initialize the volume or replace that volume with another initialized backup volume.

```
Volume not RT-11 format, Are you sure?
```

or

```
Volume contains files. Are you sure?
```

Type Y or any string beginning with Y, followed by a carriage return, to initialize the volume already mounted. BUP automatically performs the copy operation once the initialization completes. If you choose not to initialize the volume, type anything other than Y. BUP prompts you to mount another volume.

```
Mount output volume in <dev>; Continue?
```

The following example shows a large file being backed up to RX02 diskettes. BUP finds that the second diskette has not been initialized as a backup volume, and queries for initialization.

```
* DY0:=DL1:LGFIL.DAT
Mount output volume in DY0; Continue?  Y
?BUP-I-Creating volume 1
Mount output volume in DY0; Continue?  Y
?BUP-W-Not a backup volume DY0:
DY0:/BUP Initialize; Are you sure?
Volume contains files. Are you sure?  Y
?BUP-I-Bad block scan started...
?BUP-I-No bad blocks detected
?BUP-I-Creating volume 2
```

3.3.2 Volume Backup Operation (/I)

To back up an entire volume in image mode, use the /I option.

You must first initialize the output volumes, unless the output volumes are magtapes, by using the the BUP /Z option, so BUP can recognize the volumes as backup volumes and to ensure that the volumes contain no bad blocks. (The /Z option is invalid with magtapes; BUP initializes magtapes during the backup operation.) See Section 3.3.5 for details on initializing volumes for BUP operations.

Use the following syntax to perform an image mode backup operation.

output-spec = input-device/I

where:

output-spec — represents the device in which you will mount the output volumes for the backup operation, and the file specification for the backup file. You must copy to a file even when you are backing up a volume. If you specify no output file name, BUP uses the 2-letter mnemonic of the input device (for example, DL for an RL02). The default output file type is .BUP. Wildcards are invalid in the output specification.

input-device represents the device and unit number in which you will mount the volume to be backed up.

BUP copies the input volume to the first of the output volumes until the output volume is full. If the entire input volume fits on the first of the output volumes, BUP prints an error message unless the output volume is magtape:

```
?BUP-F-Enough space on one volume -- use PIP
```

When the first of the output volumes is full, BUP prompts you to mount the next output volume. BUP also tells you which volume BUP is creating so you can properly label each volume.

```
Mount output volume in <dev>; Continue? Y
?BUP-I-Creating volume n
```

BUP repeats this process until the entire input volume has been copied.

The output file's creation date recorded in the directory is the system date during the backup operation. If no system date was set, no creation date is entered in the directory.

If BUP detects an output volume that has not been initialized as a backup volume (see Section 3.3.5), BUP prints a message and allows you to either initialize the volume as a backup volume or replace that volume with an already initialized backup volume.

```
?BUP-W-Not a backup volume DEV:
<dev:>/BUP Initialize; Are you sure? Y
```

If you are using magtape for the backup volumes, BUP always attempts to initialize the output volume, and asks you to confirm the initialization.

If BUP detects a backup volume that is not a valid RT–11 volume, or a volume that already contains files, BUP prints the appropriate message and allows you to either initialize the volume or replace that volume with another initialized backup volume.

```
Volume not RT-11 format. Are you sure?
```

or

```
Volume contains files. Are you sure?
```

Type Y or any string beginning with Y to initialize the volume already mounted. BUP automatically performs the copy operation once the initialization completes. If you choose not to initialize the volume, type anything other than Y. BUP prompts you to mount another volume by printing the following message.

```
Mount output volume in <dev>; Continue?
```

The following command backs up an RL02 volume to several RX02 diskettes. The backup volumes will contain the file DL.BUP when the backup operation is complete.

```
* DY:=DL1:/I
```

3.3.3 Directory Option (/L)

Use the /L option to display on the terminal the directory of the backup volume you specify. The syntax of the command is:

device/L

where:

device	represents the volume whose directory you want to display

The listing for random-access volumes begins with the system date and the volume number of the specified backup volume. The volume number indicates that volume's position within the set of volumes that compose a single file or volume. The volume number is followed by a four-column listing of information about each volume in the set. The first column lists the volume numbers. The second column lists the name of the file, part of which resides on that volume. The third column lists the number of blocks from the file each volume contains. The last column lists the date on which the file or volume was backed up. Underneath the four columns, BUP prints the number of free blocks on the specified volume. Since this directory information is determined when you first begin a backup operation, all the predetermined backup directory information prints when you use this option even if you do not complete the backup operation.

The following command lists the backup information for backup volume 3 of the four-volume set that composes the file CAFIL.TXT.

```
* DY0:/L
23-Jan-83

VOLUME 3 OF 4

VOLUME     FILENAME          BLOCKS     DATE

 V1        CAFIL .BUP         980       23-Jan-83

 V2        CAFIL .BUP         980       23-Jan-83

 V3        CAFIL .BUP         980       23-Jan-83

 V4        CAFIL .BUP         400       23-Jan-83

1 file, 980 blocks
0 free blocks
```

For magtapes, the listing appears in the same four-column format. However, only the current system date, and information for the magtape specified, is displayed. The third column lists the total number of blocks used in the set of magtapes that compose the file or volume.

The next example shows the backup information for a magtape.

```
*MT1:/L
23-Jan-83

VOLUME     FILENAME          BLOCKS     DATE

 V1        DL1:   .BUP        27450     23-Jan-83
```

3.3.4 Restore Option (/X)

The /X option restores a file or volume from several backup volumes to a single file or volume on a regular RT–11 structured volume. Type a command with the following syntax to restore a file.

output-spec = input-spec/X

where:

output-spec	represents the device, and file name and type to which you want to restore the backup volumes. If you specify no output file, BUP uses the input file name and type. Wildcards are invalid in the output specification.
input-spec	represents the device and file to restore. Wildcards are invalid in the input specification.

Use the following command syntax to restore a volume.

output-dev = input-spec/I/X

where:

output-dev	represents the volume to which you want to copy the backup volumes. Wildcards are invalid in the output specification.
input-spec	represents the input volume (and optionally the file name under which it is stored) to restore. Wildcards are invalid in the input specification.

BUP prompts you to mount volume 1 of the set of volumes that contain the file or volume and tells you when the restore operation begins:

```
Mount input volume n in <dev>; Continue? Y
?BUP-I-Restore operation started from volume n
```

When BUP has copied all of the first input volume to the output volume, BUP prompts you to mount the next volume. BUP continues this process

until the entire set of backup volumes composing the original file or volume have been copied to a single volume. If you mount a volume with the wrong volume number, BUP prints an error message and prompts you to mount the correct volume.

```
?BUP-E-Wrong volume number
Mount input volume n in <dev>; Continue?
```

If you mount a volume with the correct volume number but it contains the wrong file, BUP notifies you and prompts you to mount the correct volume.

```
?BUP-W-File not found DEV:FILNAM.TYP
Mount input volume number n in <dev>; Continue?
```

In either case, type any string beginning with Y after you have replaced the incorrect volume with the correct volume. Type any string beginning with N to abort the entire operation. Any other response causes BUP to continue to prompt you to mount the proper volume.

The following example shows a backup file being restored from two RX02 diskettes to a single file on an RL02 disk.

```
*DL1:=DY1:LGFIL.DAT/X
Mount input volume 1 in DY1; Continue?
?BUP-I-Restore operation started from volume 1
Mount input volume 2 in DY1; Continue?
?BUP-I-Restore operation started from volume 2
?BUP-I-Copy operation is complete
```

The next command restores a volume from RX02 diskettes to an RL02 disk.

```
*DL1:=DY0:/X/I
```

3.3.5 Initialize Option (/Z)

You must use the /Z option to initialize any volume, except magtape, before you can use that volume for output during a backup operation. (The /Z option is invalid with magtape. BUP initializes magtapes during the backup operation.) The /Z option clears the directory of the volume and writes information into the home block (block 0) so BUP can recognize the volume as a backup volume. In addition, /Z scans the volume for bad blocks, since backup volumes must not contain bad blocks.

The syntax of the initialization command is as follows:

device:/Z

where:

device	represents the device that contains the volume you want to initialize

BUP prompts you to confirm the initialization. Type Y or any string beginning with Y to continue with the initialization of the backup volume. If your response begins with anything other than Y or you type CTRL/C, the operation is aborted and the CSI asterisk appears.

You can use the /Y option with /Z to suppress the confirmation message that prints during initialization.

The following command initializes a double-density diskette as a backup volume:

```
*DY0:/Z
DY0:/BUP Initialize; Are you sure? Y
?BUP-I-Bad block scan started...
?BUP-I-No bad blocks detected
```

If BUP detects bad blocks, BUP prints the following message to notify you that the volume could not be successfully initialized:

```
?BUP-I-Bad blocks detected; use another volume
```

You must mount and initialize another volume.

After BUP initializes a volume, no files exist in the directory. Therefore, if you attempt to have BUP initialize a volume that already contains files, BUP warns you that files exist and asks you to confirm the initialization.

```
Volume contains files.  Are you sure?
```

Type anything that begins with Y to continue the initialization operation. Type anything else to abort the operation.

To return a BUP-initialized volume to an RT–11 structure volume for use other than with BUP, initialize the volume using DUP. Refer to Chapter 6 of this manual for more information on initializing volumes with DUP.

Chapter 4
Directory Program (DIR)

The directory program (DIR) performs a wide range of directory listing operations. It can list directory information about a specific device, either in summarized form — where only the number of files stored per segment is given — or in more detailed form — where file names, file types, creation dates, and other file information is given. DIR can organize its listings in several ways, such as alphabetically or chronologically.

4.1 Calling and Terminating DIR

To call DIR from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
. R DIR (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you enter only a carriage return in response to the asterisk, DIR prints its current version number. You can type CTRL/C to halt DIR and return control to the monitor when DIR is waiting for input from the console terminal. You must type two CTRL/Cs to abort DIR at any other time. To restart DIR, type R DIR or REENTER in response to the monitor's dot.

4.2 Directory Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line that DIR accepts. Unless otherwise indicated, numeric arguments are interpreted as octal. Remember to put a decimal point after a decimal number to distinguish it from an octal number.

Some of the DIR options accept a date as an argument in the command line. The syntax for specifying the date is:

dd.:mmm:yy.

where:

dd.	represents the day (a decimal integer in the range 1–31)

mmm represents the month (the first three characters of the name of the month)

yy. represents the year (a decimal integer in the range 73–99)

You can specify only one input device and one output device, but you can specify up to six file names on the input device. The default device for output is the terminal. The default file type for an output file is .DIR. The default device for input is DK:. If you omit the input specification completely, DIR uses DK:*.*. If you do not supply an option, DIR performs the /L operation. Note that wildcards are valid with DIR for the input specification only.

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, DIR prints -BAD- in the creation date column of each file created beyond the end-of-month. (Note that you can eliminate a -BAD- entry by using the RENAME/SETDATE command after you have set the date.)

4.3 Reading Directory Listings

Directory listings normally print on the terminal in two columns. Read the entries across the columns, moving from left to right, one row at a time. Directory listings that are sorted, however, are an exception to this. (Sorted directories are produced by /A, /R, and /S options.) Read these listings by reading the left column from top to bottom, then reading the right column from top to bottom.

4.4 Options

You can perform many different directory operations by specifying options in the DIR command line. Table 4–1 summarizes the operations these options permit you to perform with DIR. The sections following the table describe the various DIR options and give examples; the options are arranged alphabetically in these sections.

4.4.1 Alphabetical Option (/A)

The /A option lists the directory of the device you specify in alphabetical order by file name and type. Note that /A sorts numbers after letters. It has the same effect as the /S:NAM option. The following example lists the directory of device DY0: in alphabetical order.

```
* DY0:/A
 14-Mar-83
BUILD .SAV   100  06-Sep-82    SWAP   .SYS    25  05-Dec-82
DY    .SYS     3  06-Sep-82    SYSMAC.MAC    41  19-Nov-82
MYPROG.MAC   36P  12-Oct-82    TM     .MAC    25  27-Nov-82
RFUNCT.SYS     4  19-Nov-82    TT     .SYS     2  19-Nov-82
RT11SJ.SYS    67  19-Nov-82    VTMAC .MAC      7  19-Nov-82
 10 Files, 306 Blocks
 180 Free Blocks
```

Table 4–1: DIR Options

Option	Section	Operation
/A	4.3.1	Lists the directory of the volume you specify in alphabetical order by file name and type (this is the same as /S:NAM).
/B	4.3.2	Lists the directory of the volume you specify, including file names and types, creation dates, starting block numbers, and the number of blocks in each file. For magtape, the starting block number is the file sequence number. Note that DIR lists block numbers in decimal, unless you use the /O option.
/C[:n]	4.3.3	Lists the directory in n columns; n is an integer in the range 1–9. The default value is two columns for normal listings and five columns for abbreviated listings.
/D[:date]	4.3.4	Lists a directory containing only those files having the date you specify. If you do not supply a date, DIR uses the system's current date.
/E	4.3.5	Adds unused spaces and their sizes to the listing of the volume directory.
/F	4.3.6	Prints a five-column, short directory (file names and types only) of the volume you specify.
/G	4.3.7	Lists the file you specify and all files that follow it in the directory. This option does not list any files that precede the file you specify.
/J[:date]	4.3.8	Prints a directory of the files created on or after the date you specify. If you do not supply a date, DIR uses the system's current date.
/K[:date]	4.3.9	Prints a directory of files created before the date you specify. If you do not supply a date, DIR uses the system's current date.
/L	4.3.10	Lists the directory of the volume you specify, including the number of files, their dates, and the number of blocks each file occupies. (This is the default operation.)
/M	4.3.11	Lists a directory of unused areas of the volume you specify.
/N	4.3.12	Lists a summary of the device directory.
/O	4.3.13	Similar to /L but lists the sizes and block numbers of the files in octal.
/P	4.3.14	Prints a directory of the volume you specify, excluding the files you list.
/Q	4.3.15	Lists a directory of the volume you specify, listing the file names and types, sizes, creation dates, and starting block numbers of files that have been deleted and whose file name information has not been destroyed.
/R	4.3.16	Lists the files in the reverse order of the sort specified with /A or /S.

(Continued on next page)

Table 4–1: DIR Options (Cont.)

Option	Section	Operation
/S[:xxx]	4.3.17	Lists the directory of the volume you specify in the order you specify; xxx indicates the order in which DIR sorts the listing (xxx can be DAT, NAM, POS, SIZ, or TYP).
/T	4.3.18	Lists a directory of all files on the volume you specify that are protected against deletion.
/U	4.3.19	Lists a directory of all files on the volume you specify that are not protected against deletion.
/V[:ONL]	4.3.20	Lists the volume ID and owner name as part of the directory listing header. If you specify /V:ONL, DIR lists only the volume ID and owner name.

4.4.2 Block Number Option (/B)

The /B option includes the starting block number in decimal of all the files listed in a directory of the volume you specify. The following example lists the directory of device DY0:, including the starting block numbers of files.

```
* DY0:/B
 14-Jan-83
FSM    .MAC     31P 19-Nov-82   2955   BATCH .MAC    102P 19-Nov-82   2986
ELCOPY.MAC       8P 19-Nov-82   3088   ELINIT.MAC     15P 19-Nov-82   3096
ELTASK.MAC      15P 19-Nov-82   3111   ERROUT.MAC     48P 19-Nov-82   3126
ERRTXT.MAC       9P 19-Nov-82   3174   SYCND .BL       3P 19-Nov-82   3183
SYSTBL.BL        4P 19-Nov-82   3186   SYCND .DIS      5P 19-Nov-82   3190
SYSTBL.DIS       4P 19-Nov-82   3195   SYCND .HD       5P 19-Nov-82   3199
ABSLOD.SAV      48  15-MAR-82   3204   CHESS .SAV     40  17-Aug-82   3252
PETAL .SAV      36  11-Sep-82   3292   LAMP  .SAV     29  16-Mar-82   3328
WUMPUS.SAV      30  16-Mar-82   3357
 17 Files, 348 Blocks
 138 Free blocks
```

4.4.3 Columns Option (/C[:n])

The /C[:n] option lists the directory in the number of columns you specify. The argument, n, represents an integer in the range 1–9. If you do not use the /C:n option, DIR lists the directory in two columns for normal listings and five columns for abbreviated listings. The following command, for example, lists the directory of device DY1: in one column.

```
* DY1:/C:1
 4-Jan-83
SWAP   .SYS    25P 19-Nov-82
RT11SJ.SYS    67P 19-Nov-82
RT11FB.SYS    80P 19-Nov-82
RT11BL.SYS    64P 19-Nov-82
TT     .SYS     2P 19-Nov-82
DT     .SYS     3P 19-Nov-82
DP     .SYS     3P 19-Nov-82
 7 Files, 244 Blocks
 242 Free blocks
```

4.4.4 Date Option (/D[:date])

The /D[:date] option includes in the directory listing only those files having the date you specify. The default date is the system's current date. For example, the following command lists all the files created on January 14, 1983.

```
* DY0:/D:14.:JAN:83.
 15-Jan-83
RT11SJ.SYS     67P 14-Jan-83          RT11FB.SYS     80P 14-Jan-83
RT11BL.SYS     63P 14-Jan-83          DX     .SYS     3P 14-Jan-83
SWAP   .SYS    25P 14-Jan-83          TT     .SYS     2P 14-Jan-83
DP     .SYS     3P 14-Jan-83          DY     .SYS     4P 14-Jan-83
LP     .SYS     2P 14-Jan-83          PIP    .SAV    16  14-Jan-83
DUP    .SAV    41  14-Jan-83          RESORC.SAV    15  14-Jan-83
DIR    .SAV    17  14-Jan-83          RK     .SYS     3  14-Jan-83
EDIT   .SAV    19  14-Jan-83          DD     .SYS     5  14-Jan-83
SRCCOM.SAV     13  14-Jan-83          BINCOM.SAV    11  14-Jan-83
SLP    .SAV     9  14-Jan-83          SIPP   .SAV    14  14-Jan-83
 20 Files, 412 Blocks
 73 Free blocks
```

4.4.5 Entire Option (/E)

The /E option lists the entire directory including the unused areas and their sizes in blocks (decimal). Use it to find free space before you extend a file (with the monitor CREATE command or DUP /C option). The following example lists the entire directory of device DY1:, including unused areas.

```
* DY1:/E
 20-Mar-83
SWAP   .SYS    25P 23-Oct-82          RT11SJ.SYS    67P 23-Oct-82
RT11FB.SYS     80P 19-Nov-82          RT11BL.SYS    64P 19-Nov-82
TT     .SYS     2P 19-Nov-82          DT     .SYS     3P 19-Nov-82
DP     .SYS     3P 23-Oct-82          DX     .SYS     3P 19-Nov-82
DY     .SYS     4P 19-Nov-82          RF     .SYS     3P 19-Nov-82
RK     .SYS     3P 19-Nov-82          DL     .SYS     4P 23-Oct-82
DM     .SYS     5P 23-Oct-82          DS     .SYS     3P 19-Nov-82
DD     .SYS     5P 23-Oct-82          LP     .SYS     2P 23-Oct-82
LS     .SYS     2P 19-Nov-82          CR     .SYS     3P 19-Nov-82
MS     .SYS     9P 27-Nov-82          MTHD   .SYS     3P 23-Oct-82
DISMT1.COM      9P 27-Nov-82          MMHD   .SYS     4P 19-Nov-82
NUMBER.PAS      1  11-Dec-82          TONY   .AGP    14  17-Aug-82
NUM3   .LST     1  13-Dec-82          < UNUSED >    565
 25 Files, 322 Blocks
 164 Free blocks
```

4.4.6 Fast Option (/F)

The /F option lists only file names and file types, omitting file lengths and associated dates. For example, the following command lists only file names and types from device DY0:.

```
*[ DY0:/F
 16-Aug-82
DY     .SYS    PIP    .SAV    DIR    .SAV    DUP    .SAV    SWAP .SYS
RT11SJ.SYS     RT11FB.SYS     RT11BL.SYS     TT     .SYS    DT   .SYS
 10 Files, 312 Blocks
 174 Free blocks
```

4.4.7 Begin Option (/G)

The /G option lists the directory of the volume you specify, beginning with the file you specify and including all the files that follow it in the directory.

Usually, the disk you are using as a system device contains a number of files the operating system needs. These files include .SYS monitor files, .SAV utility program files, and various .OBJ, .MAC, and .BAK files. They are generally grouped together and usually listed at the beginning of a normal volume directory. Files that you create and use, such as source files and text files, are also generally grouped together and follow the operating system files in the directory. If you specify the name of the last system file with the /G in the command line, DIR prints a directory of only those files that you created and stored on the volume.

The following command, for example, lists the last system file (CT.SYS) and all the user files that follow it.

```
* DY0:CT.SYS/G
 10-Jan-83
CT     .SYS      5  10-Aug-82          DIR    .SAV     17  03-Aug-82
RK     .SYS      3  13-Aug-82          EDIT   .SAV     19  03-Aug-82
STARTS.COM       1  27-Aug-82          DD     .SYS      5  19-Aug-82
SRCCOM.SAV      13  13-Aug-82          BINCOM.SAV      11  05-Oct-82
SLP    .SAV      9  13-Aug-82          SIPP   .SAV     14  05-Oct-82
 10 Files, 107 Blocks
 73 Free blocks
```

4.4.8 Since Option (/J[:date])

The /J[:date] option lists a directory of all files stored on the device you specify created on or after the date you supply. The default date is the system's current date. The following command lists all files on device DY0: created on or after January 20, 1983.

```
* DY0:/J:20.:JAN:83.
 20-Mar-83
RT11SJ.SYS     67P 28-Jan-83          RT11FB.SYS     80P 02-Feb-83
RT11BL.SYS     63P 19-Feb-83          DX     .SYS      3P 10-Mar-83
SWAP   .SYS    25P 02-Feb-83          TT     .SYS      2P 15-Mar-83
SIPP   .SAV    14  02-Feb-83
 7 Files, 154 Blocks
 332 Free blocks
```

4.4.9 Before Option (/K[:date])

The /K[:date] option prints a directory of files created before the date you specify. The default date is the system's current date. The following command lists all files stored on device DY1: created before March 15, 1983.

```
* DY1:/K:15.:MAR:83.
 20-Mar-83
FORTRA.SAV    191  14-Mar-83          BASIC  .SAV     51  25-Feb-83
 2 Files, 242 Blocks
 38 Free blocks
```

4.4.10 Listing Option (/L)

The /L option lists the directory of the volume you specify. The listing contains the current date, all files and their associated creation dates, the number of blocks used by each file, total free blocks on the device (if disk), the number of files listed, and the total number of blocks used by the files. File lengths, number of blocks, and number of files are indicated as decimal values. For example, the following command lists on the line printer the directory for device DY1:.

```
LP:=DY1:/L
```

The line printer output looks like this:

```
 20-Nov-82
RT11SJ.SYS     67P 03-Jul-82          RT11FB.SYS     80P 13-Aug-82
RT11BL.SYS     63P 15-Mar-82          DX     .SYS     3P 13-Aug-82
SWAP   .SYS    25P 13-Aug-82          TT     .SYS     2P 13-Aug-82
DP     .SYS     3P 13-Aug-82          DY     .SYS     4P 13-Aug-82
LP     .SYS     2P 20-Nov-82          PIP    .SAV    16  25-Jul-82
DUP    .SAV    41  26-Mar-82          RESORC.SAV    15  13-Aug-82
EDIT   .SAV    19  13-Aug-82          STARTS.COM     1  27-Aug-82
SIPP   .SAV    14  13-Aug-82
 15 Files, 413 Blocks
 73 Free blocks
```

Note that if you specify no options in the command string, this is the default directory operation.

4.4.11 Unused Areas Option (/M)

The /M option lists only a directory of unused areas and their size on the volume you specify. For example, the following command lists all the unused areas on device DL0:.

```
* DL0:/M
 14-Dec-82
< UNUSED >     11              < UNUSED >      2
< UNUSED >     26              < UNUSED >     32
< UNUSED >      1              < UNUSED >    525
< UNUSED >      0              < UNUSED >    565
 0 Files, 0 Blocks
 1162 Free blocks
```

4.4.12 Summary Option (/N)

The /N option lists a summary of the volume directory. The summary lists the number of files in each directory segment and the number of segments in use on the volume you specify. The segments are listed in the order in which they are linked on the volume.

The following command lists the summary of the directory for device DK:.

```
* /N
 14-Jan-83

    44 Files in segment 1

    46 Files in segment 4

    37 Files in segment 2

    34 Files in segment 5

    38 Files in segment 3

    16 Available segments, 5 in use

 199 Files, 3647 Blocks
 1115 Free blocks
```

4.4.13 Octal Option (/O)

The /O option is similar to the /L option, but lists the sizes (and starting block numbers if you use /B) of the files in octal. If the device you specify is a magnetic tape, DIR prints the sequence number in octal. For example, the following command lists the directory of device DY0:, with sizes in octal.

```
* DY0:/O
 14-Jan-83 Octal
MYPROG.MAC      44P 12-Nov-82        TM     .MAC      31  27-Nov-82
VTMAC .MAC       7  18-Oct-82        SYSMAC.MAC      51  19-Nov-82
SWAP  .SYS      31  05-Sep-82        ANTON .MAC       4  19-Nov-82
RT11SJ.SYS     103  19-Nov-82        TT     .SYS      2  19-Nov-82
DX    .SYS       3  29-Aug-82        BUILD .MAC     144  19-Nov-82
 10 Files, 462 Blocks
 264 Free blocks
```

4.4.14 Exclude Option (/P)

The /P option lists a directory of all files on a volume, excluding those that you specify. You may specify up to six files.

```
* DY1:_*.SAV/P
 29-Feb-83
RT11SJ.MAC     67P 06-Jan-83          RT11FB.MAC     80P 06-Jan-83
RT11BL.MAC     63P 06-Jan-83          DY     .MAC     3P 06-Jan-83
SWAP  .MAC     25P 06-Jan-83          TT     .MAC     2P 06-Jan-83
DP    .MAC      3P 06-Jan-83          DY     .MAC     4P 06-Jan-83
LP    .MAC      2P 06-Jan-83          RK     .MAC     3  06-Jan-83
STARTS.COM      1  27-Jan-83          DD     .MAC     5  06-Jan-83
 12 Files, 258 Blocks
 73 Free blocks
```

This command lists all files on device DY1: except .SAV files.

4.4.15 Deleted Option (/Q)

The /Q option lists a directory of the volume you specify, listing the file names, types, sizes, creation dates, and starting block numbers in decimal of files that have been deleted but whose file name information has not been destroyed. The file names that print represent either tentative files or files that have been deleted. This can be useful in recovering files that have been accidentally deleted. Once you identify the file name and location, you can use DUP to rename the area. See Section 6.3.1 for this procedure.

```
* DISK.DIR=/Q
```

This command creates a file called DISK.DIR on device DK: that contains directory information about unused areas from device DK:. Use the monitor TYPE command to read the file:

```
. TYPE DISK.DIR/LOG
 Files copied:
DK:DISK.DIR      to TT:
 12-Oct-82
EXAMPL.FOR     23  03-Sep-82  1403     MTHD  .SMP     5  09-Sep-82 2895
SCOPE .PIC      3  22-Sep-82  2926
 0 Files, 0 Blocks
 0 Free blocks
```

4.4.16 Reverse Option (/R)

The /R option lists a directory in the reverse order of the sort you specify with the /A or /S option.

```
* DY0:/S:SIZ/R
 14-Jan-83
BUILD .MAC    100  06-Sep-82          TM     .MAC    25  27-Nov-82
RT11SJ.SYS     67  19-Nov-82          VTMAC  .MAC     7  19-Nov-82
SYSMAC.MAC     41  19-Nov-82          RFUNCT.SYS      4  19-Nov-82
MYPROG.MAC     36P 12-Oct-82          DX     .SYS     3  06-Sep-82
SWAP  .SYS     25  05-Dec-82          TT     .SYS     2  19-Nov-82
 10 Files, 306 Blocks
 180 Free blocks
```

This command lists the directory of device DY0: in reverse file size order (from largest to smallest).

4.4.17 Sort Option (/S[:xxx])

The /S[:xxx] option sorts the directory of the specified volume according to a three-character code you specify as :xxx. Table 4–2 summarizes the codes and their functions.

Table 4-2: Sort Codes

Code	Function
DAT	Chronological by creation date. Files that have the same date are sorted alphabetically by file name and file type.
NAM	Alphabetical by file name. Files that have the same file name are sorted alphabetically by file type (this has the same effect as the /A option).
POS	According to the position of the files on the device. This is the same as using /S with no code.
SIZ	Based on file size (in blocks). Files that are the same size are sorted alphabetically by file name and file type. Files are sorted from smallest to largest unless you also use /R.
TYP	Alphabetical by file type. Files that have the same file type are sorted alphabetically by file name.

The following examples illustrate the /S option.

```
* DY0:/S:DAT
 4-Feb-83
BUILD .MAC     100  06-Sep-82          SYSMAC.MAC     41  19-Nov-82
DY    .SYS       3  06-Sep-82          TT    .SYS      2  19-Nov-82
MYPROG.MAC      36P 12-Oct-82          VTMAC .MAC      7  19-Nov-82
RFUNCT.MAC       4  19-Nov-82          TM    .MAC     25  27-Nov-82
RT11SJ.SYS      67  19-Nov-82          SWAP  .SYS     25  05-Dec-82
 10 Files, 306 Blocks
 180 Free blocks

* DY0:/S:NAM
 4-Feb-83
BUILD .MAC     100  06-Sep-82          SWAP  .SYS     25  05-Dec-82
DY    .SYS       3  06-Sep-82          SYSMAC.MAC     41  19-Nov-82
MYPROG.MAC      36P 12-Oct-82          TM    .MAC     25  27-Nov-82
RFUNCT.SYS       4  19-Nov-82          TT    .SYS      2  19-Nov-82
RT11SJ.SYS      67  19-Nov-82          VTMAC .MAC      7  19-Nov-82
 10 Files, 306 Blocks
 180 Free Blocks

* DY0:/S:POS
 4-Feb-83
RT11SJ.SYS      67  19-Nov-82          BUILD .MAC    100  06-Sep-82
DY    .SYS       3  06-Sep-82          SYSMAC.MAC     41  19-Nov-82
MYPROG.MAC      36P 12-Oct-82          TM    .MAC     25  27-Nov-82
SWAP  .SYS     25  05-Dec-82           VTMAC .MAC      7  19-Nov-82
RFUNCT.SYS       4  19-Nov-82          TT    .SYS      2  19-Nov-82
 10 Files, 306 Blocks
 180 Free blocks

* DY0:/S:SIZ
 4-Jan-83
TT    .SYS       2  19-Nov-82          TM    .MAC     25  27-Nov-82
DY    .SYS       3  06-Sep-82          MYPROG.MAC     36P 12-Oct-82
RFUNCT.SYS       4  19-Nov-82          SYSMAC.MAC     41  19-Nov-82
VTMAC .MAC       7  19-Nov-82          RT11SJ.SYS     67  19-Nov-82
SWAP  .SYS     25  05-Dec-82           BUILD .MAC    100  06-Sep-82
 10 Files, 306 Blocks
 180 Free blocks
```

```
*LDY0:/S:TYP
 14-Dec-82
BUILD .MAC     100  06-Sep-82          DY     .SYS      3  06-Sep-82
MYPROG.MAC      36P 12-Oct-82          RFUNCT.SYS       4  19-Nov-82
SYSMAC.MAC      41  19-Nov-82          RT11SJ.SYS      67  19-Nov-82
TM     .MAC      25  27-Nov-82          SWAP   .SYS     25  05-Dec-82
VTMAC .MAC        7  19-Nov-82          TT     .SYS      2  19-Nov-82
 10 Files, 306 Blocks
 180 Free blocks
```

4.4.18 Protection Option (/T)

The /T option includes in the directory listing only those files on the volume you specify that are protected against deletion. A letter P next to the block size number in the file's directory entry indicates that the file is protected. The following command lists only those files on DK: that are protected.

```
* DK:/S:SIZ/R/T
 5-Jan-83
BUILD .MAC          100P 06-Sep-82     TM     .MAC     25P 27-Nov-82
RT11SJ               67P 19-Nov-82     VTMAC .MAC       7P 19-Nov-82
SYSMAC.MAC           41P 19-Nov-82     RFUNCT.SYS       4P 19-Nov-82
MYPROG.MAC           36P 12-Oct-82     DX     .SYS      3P 06-Sep-82
SWAP   .SYS          25P 05-Dec-82     TT     .SYS      2P 19-Nov-82
 10 Files, 306 Blocks
 5584 Free blocks
```

4.4.19 No Protection Option (/U)

The /U option includes in the directory listing only those files on the volume you specify that are not protected against deletion. Files that are not protected do not have a P in the file's directory entry. The following command lists only those files on DK: that are not protected.

```
* /S:SIZ/R/U
 14-Dec-82
COUNT .MAC          100 06-Sep-82      SBT .TXT         25 27-Nov-82
ASCII .MAC           67 19-Nov-82      MAIL.MAI          7 19-Nov-82
SUBONE.MAC           41 19-Nov-82      SQRT.FOR          4 19-Nov-82
MYPROG.MAC           36 12-Oct-82      DX   .SYS         3 06-Sep-82
 8 Files, 283 Blocks
 325 Free blocks
```

4.4.20 Volume ID Option (/V[:ONL])

The /V option prints the volume identification and owner name as part of the directory listing header. The optional argument, :ONL, prints only the volume ID and owner name. You can combine /V with any other option.

The following example uses the /V option.

```
* DY:/V
 14-Jan-83
 Volume ID: BACKUP2
 Owner    : Marcy
SWAP   .SYS     25P 19-Nov-82          RT11SJ.SYS     67P 19-Nov-82
RT11FB.SYS      80P 19-Nov-82          RT11BL.SYS     64P 19-Nov-82
TT     .SYS      2P 19-Nov-82          DT     .SYS     3P 19-Nov-82
DP     .SYS      3P 19-Nov-82          DX     .SYS     3P 19-Nov-82
DY     .SYS      4P 19-Nov-82          RF     .SYS     3P 19-Nov-82
RK     .SYS      3P 19-Nov-82          DL     .SYS     4P 19-Nov-82
 12 Files, 271 Blocks
 215 Free blocks
```

The next example uses the :ONL argument.

```
* DY0:/V:ONL
Volume ID: RT11 V5
Owner    : Donna
```

Chapter 5
DUMP Program (DUMP)

The DUMP program prints on the console or line printer, or writes to a file, all or any part of a file as words or bytes (in octal), ASCII characters, or Radix–50 characters. DUMP is particularly useful for examining directories and files that contain binary data.

5.1 Calling and Terminating DUMP

To call the DUMP program from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
.R DUMP (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. If you respond to the asterisk by typing only a carriage return, DUMP prints its current version number.

You can type CTRL/C to halt DUMP and return control to the monitor when DUMP is waiting for input from the console terminal. You must type two CTRL/Cs to abort DUMP at any other time. To restart DUMP, type R DUMP or REENTER in response to the monitor's dot.

5.2 DUMP Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line that DUMP accepts. If you do not specify an output file, the listing prints on the line printer. If you do not specify a file type for an output file, the system uses .DMP.

5.3 Options

Table 5–1 summarizes the options that are valid for DUMP.

Table 5-1: DUMP Options

Option	Function
/B	Outputs bytes, in octal.
/E:n	Ends output at block n, where n is an octal block number.
/G	Ignores input errors.
/N	Suppresses ASCII output.
/O:n	Outputs only block n, where n is an octal block number. With this option, you can dump only one block for each command line.
/S:n	Starts output with block n, where n is an octal block number. For random-access devices, n may not be greater than the number of blocks in the file.
/T	Defines a tape as non-file-structured.
/W	Outputs words, in octal (the default mode).
/X	Outputs Radix–50 characters.

ASCII characters are always dumped unless you type /N.

If you specify an input file name, the block numbers (n) you supply are relative to the beginning of that file. If you do not specify a file name, that is, if you are dumping a device, the block numbers are the absolute (physical) block numbers on that device. Remember that the first block of any file or device is block 0.

NOTE

DUMP does not print data from track 0 of diskettes.

DUMP handles operations that involve magtape differently from operations involving random-access devices.

If you dump an RT–11 file-structured tape and specify only a device name in the input specification, DUMP reads only as far as the logical EOF1. Logical end-of-tape is indicated by an end-of-file label followed by two tape marks. For non-file-structured tape, logical end-of-tape is indicated by two consecutive tape marks. For magtape dumps, tape mark messages appear in the output listing as DUMP encounters them on the tape.

If you use /S:n with magtape, n can be any positive value. However, an error can occur if n is greater than the number of blocks written on the tape. For example, if a tape has 100 written blocks and n is 110, an error can occur if DUMP accesses past the 100th block. If you specify /E:n, DUMP reads the tape from its starting position (block 0, unless you specify otherwise) to block n or to logical end-of-tape, whichever comes first.

5.4 Example Commands and Listings

This section includes sample DUMP commands and the listings they produce.

The following command string directs DUMP to print, in words, information contained in block 1 of the file DMPX.SAV stored on device DK:.

```
*DMPX.SAV/O:1

DMPX.SAV/O:1
BLOCK NUMBER 000001
000/ 000000 042062 000000 000000 000000 000000 000000 000000 *..2D............*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
040/ 000000 000000 001002 000000 000000 003356 001010 001104 *..........n...D.*
060/ 000014 000400 004001 000000 000000 000000 000000 000000 *................*
100/ 000000 000000 045504 043072 046111 030505 044456 046523 *....DK:FILE1.ISM*
120/ 000000 046061 000000 000000 000000 000000 000000 000000 *..1L............*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
160/ 000000 000000 000000 000000 000000 001356 002000 001234 *..........n.....*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
360/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
```

In the printout above, the heading shows which block of the file follows. The numbers in the leftmost column indicate the byte offset from the beginning of the block. Remember that these are all octal values and that there are two bytes per word. The words that were dumped appear in the next eight columns. The rightmost column contains the ASCII equivalent of each word. DUMP substitutes a dot (.) for nonprinting codes, such as those for control characters.

The next command dumps block 1 of file PIP.SAV. The /N option suppresses ASCII output.

```
* PIP.SAV/N/O:1

SY:PIP.SAV/N/O:1
BLOCK NUMBER  000001
000/ 060502 010046 010146 010246 000422 062701 001100 012102
020/ 022512 001406 012100 005046 011146 010246 104217 103405
040/ 012602 012601 012600 011505 000205 104376 175400 012767
060/ 011501 177724 016701 000012 005021 020167 000006 103774
100/ 000743 005562 015260 005562 000006 002112 005562 000013
120/ 003147 014100 000022 000211 014100 000023 000423 014100
140/ 000025 000470 004537 001002 000006 006456 004537 001002
160/ 000014 005764 004537 001002 000022 014102 004537 001002
200/ 000030 014102 004537 001002 000036 014120 000000 000000
220/ 000000 000000 000000 000000 000000 000000 000000 000000
240/ 000000 000000 000000 000000 000000 000000 000000 000000
260/ 000000 000000 000000 000000 000000 000000 000000 000000
300/ 000000 000000 000000 001000 015260 000000 000000 000000
320/ 000000 000000 000000 000000 000000 000000 000000 000000
340/ 000000 000000 000000 000000 000000 000000 000000 000000
360/ 000000 000000 000000 000000 000000 000000 000000 000000
400/ 000000 000000 000000 000000 000000 000000 000000 000000
420/ 000000 000000 002056 002063 003452 000000 005020 000000
440/ 000000 000000 000000 000000 000000 000000 000000 000000
460/ 000000 000000 000000 000000 000000 000000 000000 000000
500/ 000000 000000 000000 000000 000000 000000 000000 000000
520/ 000000 000000 000000 000000 000000 000000 000000 000000
540/ 000000 000000 000000 000000 000000 000000 000000 000000
560/ 000000 000000 000000 000000 000000 000000 000000 000000
600/ 000000 000000 000000 000000 000000 000000 000000 000000
620/ 000000 000000 000000 000000 000000 000000 000000 000000
640/ 000000 000000 000000 000000 000000 000000 000000 000000
660/ 000000 000000 000000 000000 000000 000000 000000 000000
700/ 000000 000000 000000 000000 000000 000000 000000 000000
720/ 000000 000000 000000 000000 000000 000000 000000 000000
740/ 000000 000000 000000 000000 000000 000000 000000 000000
760/ 000000 000000 000000 000000 000000 000000 000000 000000
```

The following command dumps block 1 of SYSMAC.MAC in bytes. ASCII equivalents appear underneath each byte.

```
* SYSMAC.MAC/B/O:1

SY:SYSMAC.MAC/B/O:1
BLOCK NUMBER  000001
000/ 120 040 117 106 040 040 124 110 105 040 040 123 117 106 124 127
     P       O   F           T   H   E           S   O   F   T   W
020/ 101 122 105 040 040 111 123 040 040 110 105 122 105 102 131 015
     A   R   E           I   S           H   E   R   E   B   Y   .
040/ 012 073 040 124 122 101 116 123 106 105 122 122 105 104 056 015
     .   ;       T   R   A   N   S   F   E   R   R   E   D   .   .
060/ 012 073 015 012 073 040 124 110 105 040 111 116 106 117 122 115
     .   ;   .   .   ;       T   H   E       I   N   F   O   R   M
100/ 101 124 111 117 116 040 111 116 040 124 110 111 123 040 123 117
     A   T   I   O   N       I   N       T   H   I   S       S   O
120/ 106 124 127 101 122 105 040 111 123 040 123 125 102 112 105 103
     F   T   W   A   R   E       I   S       S   U   B   J   E   C
140/ 124 040 124 117 040 103 110 101 116 107 105 040 040 127 111 124
     T       T   O       C   H   A   N   G   E           W   I   T
```

```
160/ 110 117 125 124 040 040 116 117 124 111 103 105 015 012 073 040
     H   O   U   T           N   O   T   I   C   E   .   .   ;   .
200/ 101 116 104 040 040 123 110 117 125 114 104 040 040 116 117 124
     A   N   D           S   H   O   U   L   D           N   O   T
220/ 040 040 102 105 040 040 103 117 116 123 124 122 125 105 104 040
             B   E           C   O   N   S   T   R   U   E   D
240/ 040 101 123 040 040 101 040 103 117 115 115 111 124 115 105 116
         A   S           A       C   O   M   M   I   T   M   E   N
260/ 124 040 102 131 040 104 111 107 111 124 101 114 040 105 121 125
     T       B   Y       D   I   G   I   T   A   L       E   Q   U
300/ 111 120 115 105 116 124 015 012 073 040 103 117 122 120 117 122
     I   P   M   E   N   T   .   .   ;       C   O   R   P   O   R
320/ 101 124 111 117 116 056 015 012 073 015 012 073 040 104 111 107
     A   T   I   O   N   .   .   .   ;   .   .   ;       D   I   G
340/ 111 124 101 114 040 101 123 123 125 115 105 123 040 116 117 040
     I   T   A   L       A   S   S   U   M   E   S       N   O
360/ 122 105 123 120 117 116 123 111 102 111 114 111 124 131 040 106
     R   E   S   P   O   N   S   I   B   I   L   I   T   Y       F
400/ 117 122 040 124 110 105 040 125 123 105 040 117 122 040 040 122
     O   R       T   H   E       U   S   E       O   R           R
420/ 105 114 111 101 102 111 114 111 124 131 040 040 117 106 040 040
     E   L   I   A   B   I   L   I   T   Y           O   F
440/ 111 124 123 015 012 073 040 123 117 106 124 127 101 122 105 040
     I   T   S   .   .   ;       S   O   F   T   W   A   R   E
460/ 117 116 040 105 121 125 111 120 115 105 116 124 040 127 110 111
     O   N       E   Q   U   I   P   M   E   N   T       W   H   I
500/ 103 110 040 111 123 040 116 117 124 040 123 125 120 120 114 111
     C   H       I   S       N   O   T       S   U   P   P   L   I
520/ 105 104 040 102 131 040 104 111 107 111 124 101 114 056 015 012
     E   D       B   Y       D   I   G   I   T   A   L   .   .   .
540/ 104 056 115 101 103 122 117 040 056 056 126 061 056 056 015 012
     .   .   M   A   C   R   O       .   .   V   1   .   .   .   .
560/ 056 115 103 101 114 114 011 056 056 056 103 115 060 054 056 056
     .   M   C   A   L   L   .   .   .   .   C   M   0   ,   .   .
600/ 056 103 115 061 054 056 056 056 056 103 115 062 054 056 056 056 103
     .   C   M   1   ,   .   .   .   .   C   M   2   ,   .   .   .   C
620/ 115 063 054 056 056 056 103 115 064 054 056 056 056 103 115 065
     M   3   ,   .   .   .   C   M   4   ,   .   .   .   C   M   5
640/ 054 056 056 056 103 115 066 015 012 056 056 056 126 061 075 061
     ,   .   .   .   C   M   6   .   .   .   .   .   V   1   =   1
660/ 015 012 056 105 116 104 115 015 012 015 012 056 115 101 103 122
     .   .   .   E   N   D   M   .   .   .   .   .   M   A   C   R
700/ 117 040 056 056 126 062 056 056 015 012 056 115 103 101 114 114
     O       .   .   V   2   .   .   .   .   .   M   C   A   L   L
720/ 011 056 056 056 103 115 060 054 056 056 056 103 115 061 054 056
     .   .   .   .   C   M   0   ,   .   .   .   C   M   1   ,   .
740/ 056 056 103 115 062 054 056 056 056 103 115 063 054 056 056 056
     .   .   C   M   2   ,   .   .   .   C   M   3   ,   .   .   .
760/ 103 115 064 054 056 056 056 103 115 065 054 056 056 056 103 115
     C   M   4   ,   .   .   .   C   M   5   ,   .   .   .   C   M
```

The next example shows block 6 (the directory) of device RK0:. The output is in octal words in Radix–50 equivalents below each word.

```
*RK0:/N/X/O:6

RK0:/N/X/O:6
BLOCK NUMBER  000006
000/ 000020 000002 000004000000 000046 002000 075131 062000
       P      B      D             8     YX    SWA     P
020/ 075273 000031 000000027147 002000 071677 142302 075273
     SYS      Y           GP9     YX    RT1    1SJ    SYS
040/ 000103 000000 027147002000 071677 141262 075273 000120
      A$            GP9    XY    RT1    1FB    SYS     B
```

```
060/ 000000 027147 002000071677 141034 075273 000100 000000
            GP9     YX   RT1     1BL    SYS     AX
100/ 027147 002000 100040000000 075273 000002 000000 027147
     GP9     YX     TT            SYS     B           GP9
120/ 002000 016040 000000075273 000003 000000 027147 002000
      YX     DT           SYS     C            GP9     YX
140/ 015600 000000 075273000003 000000 027147 002000 016300
     DP             SYS     C            GP9     YX     DX
160/ 000000 075273 000003000000 027147 002000 016350 000000
            SYS     C           GP9     YX     DY
200/ 075273 000004 000000027147 002000 070560 000000 075273
     SYS     D           GP9     YX     RF           SYS
220/ 000003 000000 027147002000 071070 000000 075273 000003
       C            GP9     YX     RK           SYS     C
240/ 000000 027147 002000015340 000000 075273 000004 000000
            GP9     YX   DL            SYS     D
260/ 027147 002000 015410000000 075273 000005 000000 027147
     GP9     YX     DM            SYS     E           GP9
300/ 002000 015770 000000075273 000003 000000 027147 002000
      YX     DS           SYS     C            GP9     YX
320/ 014640 000000 075273000005 000000 027147 002000 046600
     DD             SYS     E            GP9     YX     LP
340/ 000000 075273 000002000000 027147 002000 046770 000000
            SYS     B           GP9     YX     LS
360/ 075273 000002 000000027147 002000 012620 000000 075273
     SYS     B           GP9     YX     CR           SYS
400/ 000003 000000 027147002000 052070 000000 075273 000011
       C            GP9     YX     MS           SYS     I
420/ 000000 027547 002000052150 014400 075273 000003 000000
            GWO     YX   MTH     D      SYS     C
440/ 027147 002000 015173052177 012445 000011 000000 027547
     GP9     YX     DIS   MT1     COM     I           GWO
460/ 002000 051520 014400075273 000004 000000 027147 002000
      YX     MMH     D     SYS     D            GP9     YX
500/ 015173 052200 012445000010 000000 027547 002000 052100
     DIS     MT2     COM     H            GWO     YX     MSH
520/ 014400 075273 000004000000 027147 002000 054540 000000
     D      SYS     D           GP9     YX     NL
540/ 075273 000002 000000027147 002000 062170 000000 075273
     SYS     B           GP9     YX     PC           SYS
560/ 000002 000000 027147002000 062240 000000 075273 000003
       B            GP9     YX     PD           SYS     C
600/ 000000 027147 002000012740 000000 075273 000005 000000
            GP9     YX   CT            SYS     E
620/ 027147 002000 006250000000 075273 000007 000000 027147
     GP9     YX     BA            SYS     G           GP9
640/ 002000 016130 000000073376 000051 000000 027147 002000
      YX     DUP           SAV     AA           GP9     YX
660/ 023752 050574 073376000023 000000 027147 002000 070533
     FOR     MAT     SAV     S            GP9     YX     RES
700/ 060223 073376 000017000000 027147 002000 015172 000000
     ORC     SAV     O           GPO     YX     DIR
720/ 073376 000021 000000027147 002000 075273 050553 074324
     SAV     Q           GPO     YX     SYS     MAC     SML
740/ 000052 000000 027147002000 017751 076400 073376 000023
      AB           GP9     YX     EDI     T      SAV     S
760/ 000000 027147 002000042614 000000 073376 000073 000000
            GP9     YX   KED            SAV     AS
```

Chapter 6
Device Utility Program (DUP)

The device utility program (DUP) is a device maintenance program that creates files on file-structured RT–11 devices (disks, single- and double-density diskettes, DECtape II, and magtape). It can also extend files on certain file-structured devices (disks, single- and double-density diskettes, and DECtape II), and it can compress, image copy, initialize, or boot RT–11 file-structured devices. DUP does not operate on non-file-structured devices (line printer, terminal).

6.1 Calling and Terminating DUP

To call DUP from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
.R DUP (RET)
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return at this point, DUP prints its current version number and prompts you again for a command string. You can type CTRL/C to halt DUP and return control to the monitor when DUP is waiting for input from the console terminal. You must type two CTRL/Cs to abort DUP at any other time. Note that the /S, /T, and /C operations lock out the CTRL/C command until the operation completes; these three operations cannot be interrupted with CTRL/C. To restart DUP, type R DUP or REENTER in response to the monitor's dot.

6.2 DUP Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line that DUP accepts. DUP accepts only one input file specification and one output file specification in the command line.

6.3 Options

Certain options are available for use with DUP. These options are divided into two categories: action, and mode. Action options cause specific operations to occur. You can use these options alone or with valid mode options.

Usually, you can specify only one action option at a time. Mode options modify action options. Table 6–1 illustrates which mode options you can use with a particular action option.

Table 6–1: DUP Option Combinations

Action	Mode
C	W, Y, G
D	W, Y
I	Y, G, E, F, H, R
K	W, F, G, E, F, H, R
O	Q, W, Y
S	W, X, Y
T	W, Y
U	W, Y
V	W, Y
Z	W, B, N, R, V, Y, D

Note that /V can be either an action or a mode option, depending on how you use it.

You can use DUP action options to perform operations such as creating files, copying devices, scanning for bad blocks, performing a bootstrap operation, and initializing volumes. You can use the DUP mode options to modify the action options, where necessary.

The following sections describe the various DUP options and give examples of typical uses. Table 6–2 summarizes the options you can use with DUP.

6.3.1 Create Option (/C[/G:n])

The /C option creates a file with a specific name, location, and size on the random-access device that you specify. This option is useful in recovering files that have been deleted, and for creating files to assign as logical disks (see Chapter 9). The /C option creates only a directory entry for the file. It does not store any data in the file. You must specify both the file name and file type of the file to be created.

The syntax of the command is:

filespec[n] = /C[/G:n]

where:

filespec[n] represents the device, file name, and file type of the file to be created; [n] is a decimal number representing the size in blocks of the file to be created. Note that the brackets here are part of the command; that is, they do not indicate n is optional. If you do not specify this number, DUP creates a one-block file.

Table 6–2: DUP Options

Option	Section	Function
/B[:RET]	6.3.13.4	Use with /Z to write files with the file type .BAD over any bad blocks DUP finds on the disk to be initialized. Use :RET to retain through initialization all .BAD entries created by a previous /B.
/C	6.3.1	Creates a file on the volume you specify. DUP creates the file in the first available location, unless you specify a starting block number by using the /G option.
/D	6.3.13.5	Use with /Z to uninitialize (restore) a device. Use only if no files have been transferred to the device since it was initialized.
/E:n	6.3.2	Specifies the ending block number for a read operation (used with the /I and /K options).
/F	6.3.4	Use with the /K option to transfer the file name containing the bad block together with the relative block number of the bad block in the file. Or use with /I either to copy a file to an output device or copy a device to an output file.
/G:n	6.3.1	Specifies the starting block number for a read operation (on an input device) and the starting block number for a write operation (on an output device); n is an integer that represents a block number. Use this option with the /C, /I, and /K options.
/H	6.3.2	Use with the /I option to verify that the output is equal to the input.
/I	6.3.2	Copies the image of a disk to another disk or magtape or from magtape to disk. Use with /G and /E if you want to specify block numbers.
/K	6.3.3	Scans a device for bad blocks and outputs the octal address of the bad blocks to the output device. Use with /G and /E if you want to specify block numbers as boundaries for the scan.
/N:n	6.3.13.1	Use with /Z to set the number of directory segments you require if you do not want the default size; n is an integer in the range 1–37 (octal).
/O	6.3.5	Boots the device or file you specify.
/Q	6.3.6	Use with /O to boot a volume that is not RT–11, or is a pre–Version 4 volume of RT–11.
/R[:RET]	6.3.13.3	Use with /Z to scan a device that supports bad block replacement for bad blocks, or with /I to preserve the output volume's bad block replacement table. When used with /Z, /R creates a replacement table on the disk for any bad blocks DUP finds. If you use /R:RET with /Z, DUP retains the replacement table that is already on the disk and does not prescan the disk for bad blocks.

(Continued on next page)

Table 6–2: DUP Options (Cont.)

Option	Section	Function
/S	6.3.7	Compresses a disk onto itself or onto another disk; the output device, if any, must be initialized.
/T:n	6.3.8	Extends an existing file by the number of blocks that n indicates.
/U[:xx]	6.3.9	Writes the bootstrap portion of the monitor file in blocks 0 and 2–5 of the target device. The optional argument, xx, represents the target system device name.
/V[:ONL]	6.3.10	Prints the user ID and owner name. Use it with /Z (as a mode option) to place a new user ID and owner name in block 1 of the initialized disk, or in the VOL1 header block on magtape. Using /V:ONL with /Z changes only the ID and owner name, and does not initialize the device (not applicable for magtape).
/W	6.3.11	Use with any action option except /I (but only one) to initiate an operation and then pause to allow you to change volumes. This is useful on small, single-disk systems because it lets you replace the system device with another disk before performing an operation.
/X	6.3.7	Use with /S to inhibit automatic booting of the system device when it is compressed.
/Y	6.3.12	Use with /C, /I, /O, /S, /T, U, V, or /Z to ensure immediate execution of the operation by inhibiting the confirmation messages.
/Z[:n]	6.3.13	Initializes the directory of the device you specify. The size of the directory defaults to the standard RT–11 size; use n to allocate extra directory words for each entry beyond the default.

/G:n represents the octal numeric value of the starting block of the file to be created. If you do not use /G:n, DUP creates the file in the first unused area large enough to contain the file. Use a decimal point with n (n.) to specify a decimal starting block number.

You can use the /C option to cover bad blocks on a disk by creating a file with a file type .BAD to cover the bad area.

Use /C to recover accidentally deleted files. In this case, use DIR to obtain a listing of the device. Use the /E and /Q options in DIR to list files, tentative files, empty areas, and the sizes of all areas. You can then assign a file name to the area that contains the data you lost.

You can also use DUP to set aside a file on a disk without performing any input or output operations on the file.

When you use the /C option, make sure that the area in which the file is to be created is empty (using the DIR /E and /Q options). If there are more blocks in the empty area than the file you are creating needs, DUP attempts to put the extra blocks in empty areas that are contiguous to the file you are creating. If there is not enough room in contiguous empty areas, the error message ?DUP-F-No room for file DEV:FILNAM.TYP prints, and DUP does not create the file.

The /C option checks for duplicate file names. If the file name you specify already exists on the device, DUP issues an error message and does not create a second file with the same name.

If you attempt to create a file over a tentative file (one that was opened but never closed) and the foreground is loaded, the system prompts you to confirm the operation. If you type Y to continue, DUP writes over the tentative file. Be sure that you do not write over a tentative file being used by the foreground job; this will corrupt the file and cause unpredictable results.

The following example uses /C to create a file named FILE.MAC consisting of blocks 140, 141, and 142 on device DK1:.

```
.DK1:FILE.MAC[3]=/C/G:140
```

6.3.2 Image Copy Option (/I)

The /I option copies block for block from one volume to another. This operation is applicable for magtape only when copying to or from a random-access volume, such as disk or diskette. The /I option is often used to copy one disk to another without changing the file structure or location of files on the device. For this purpose, it is an added convenience that you do not have to copy a boot block to the device. You can also copy disks that are not in RT–11 format, if they have no bad blocks. If DUP encounters a bad block on either the input or output volume, it retries the operation and performs the copy one block at a time. If no error message prints, you can assume that the transfer completed correctly.

Qualifiers to the /I option let you:

1. Specify the blocks to be read from the input device and a starting block number for the write operation on the output device.
2. Copy a file to a device, or a device to a file, by specifying a file name with either the input or output device. For example, you can copy a diskette to a file on an RL02, or a file on an RL02 to a diskette.
3. Preserve the output volume's bad block replacement table when you are copying between like volumes that support bad block replacement.
4. Verify that the output matches the input after a copy operation.

NOTE

When you use /I in an operation involving magtape, you must specify a file name and follow it with the /F option.

The syntax of the command is:

output-device: filename [/F][/G:rn] = input-device[filename]/I[/G:rn/E:rn][/F][/H][/R]
*

where:

filename — represents the file name to which you are copying the input device, or (when specified with the input device) represents the input file name you are copying to the output device. You must specify a file name when you use the /F option. If you specify an input file and you do not use /F, use the dummy file name * with the output specification. Note that you can use a file name with either the input or output, but never with both.

* — represents a dummy file name (required when you do not use the /F option, and when the output device is not a magtape). Note that either filename or *, but not both, can be specified with the output device.

/G:rn — when specified with the output device, represents the starting block number for the write operation. When specified with the input device, it represents the starting block number of the read operation.

/E:rn — represents the ending block number on the input device for the read operation.

/F — indicates that you want to copy a file to an output volume, or that you are copying an entire input volume to an output file. You must also use the /F option when you specify magtape as the input or output device (because you must always specify a file on the magtape).

/H — verifies that the input matches the output.

/R — preserves the output volume's bad block replacement table. DUP copies all blocks from the input volume to the output volume except those blocks that contain the input volume's bad block replacement table.

The command string must include an input and an output specification; there is no default device.

If one device is smaller than the other, DUP copies only the number of blocks of the smaller device. DUP may therefore copy the entire directory of the input volume, but not all of its files. If you copy a larger device to a smaller one, DUP asks you to confirm the copy operation before DUP performs the operation. If you use the /G:n and /E:n options, DUP asks you to confirm the copy only if the number of blocks to be copied is larger than the area on the output volume defined by the /G:n option and the end of the output volume.

DUP prints the confirmation message after the normal copy confirmation and after any mount messages which may be necessary if the /W option is also used. The following example shows DL0:, a large volume, being copied to DY1:, a smaller volume:

```
.R DUP
*DY1:=DL0:/I/W
DY1:/Copy; Are you sure? Y
Mount input volume in DL0: Continue? Y
Mount output volume in DY1: Continue? Y
Output device too small; Are you sure? Y
*
```

If the /F option is used the relative sizes of the input and output volumes are ignored and you are not asked to confirm the copy. The confirmation messages can also be suppressed by using the /Y option.

You can use the /H option with /I to verify that the input matches the output after an image mode copy operation.

NOTE

> The /I option does not copy track 0 of diskettes. However, this restriction has no impact on any copy operations involving RT–11 formatted diskettes.

The following examples use the /I option. The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

```
*DL1:*=DL0:/I
DL1:/Copy; Are you sure?
```

The command shown above copies all blocks from DL0: to DL1:.

```
*DL1:*/G:501=DL0:/I/G:0/E:500
DL1:/Copy; Are you sure?Y
```

The command shown above copies blocks 0–500 from DL0: to blocks 501–1000 on DL1:

```
*DL1:FLOPPY.BAK/F=DY0:/I
DL1:/Copy; Are you sure?Y
```

The last command copies device DY0: to a file named FLOPPY.BAK on DL1:.

6.3.3 Bad Block Scan Option (/K)

Some mass storage volumes (disks, diskettes, and DECtape II) have bad blocks, or they develop bad blocks as a result of age and use. You can use the

/K option to scan a device and locate bad blocks on it. DUP prints the absolute block number of those blocks that return hardware errors when DUP tries to read them. If you specify an output file, DUP prints the bad block report in that file. Remember that block numbers are octal and the first block on a device is block 0. If DUP finds no bad blocks, it prints an informational message. A complete scan of a volume takes from one to several minutes depending on the size of the volume. It does not destroy data that is stored on the device.

You can scan selected portions of a device by specifying beginning and ending block numbers. The syntax of this command is:

[filespec =]input-device:/K[/G:m][/E:n]

where:

filespec	represents the output file specification for the bad block report. If no bad blocks are found, no file is created.
/G:m	represents the block number of the first block to be scanned.
/E:n	represents the block number of the last block to be scanned.

If you specify only a starting block number, DUP scans from the block you specify to the end of the device.

If the device to be scanned has files on it, you can use /F with the /K option to print the name of the file containing the bad block and the relative block number within the file that is bad.

The following command scans the entire diskette in DY1:.

```
* DY1:/K
```

The next command scans blocks 100 to 200(decimal) of the diskette in DY1: and sends the bad block report to DY0:BLOCKS.BAD.

```
* DY0:BLOCKS.BAD=DY1:/K/G:100./E:100.
```

Sometimes a block that is reported as bad can recover. To verify whether the reported bad blocks are the result of soft or hard errors (that is, whether a bad block can recover), perform a second bad block scan and compare the two reports. Blocks reported as bad on both reports are caused by hard errors and cannot recover. Blocks that are reported as bad on the first report but not on the second report indicate that a soft error has occurred, and the blocks have recovered.

6.3.4 File Option (/F)

The file option serves two different purposes as a mode option, depending on whether you use it with /K or with /I.

When you use /F with /K, DUP does a bad block scan and displays a file name for each bad block it finds. DUP then prints a list of these bad block files along with their locations within the file. This list includes the relative block number of each bad block within the file and a report on whether each bad block is hard or soft. An example of such a list, along with the command line that generated it, follows.

```
*DY0:/K/F
    Block     Type                File          Block
000717     463.  Hard          NUMBER.PAS  000546     358.
000725     469.  Hard          ANTONY.MAC  000554     364.
000732     474.  Hard          CAESAR.MAC  000561     369.
000743     483.  Hard          < UNUSED >  000572     378.
000751     489.  Hard          < UNUSED >  000600     384.
000754     492.  Hard          < UNUSED >  000603     387.
?DUP-W-Bad blocks detected 6.
```

DUP outputs the following list if you use /F with /K on a disk that supports bad block replacement. In the column marked Type, DUP lists whether the bad block is replaced in the manufacturer's bad block replacement table or if it is hard or soft.

```
*DM1:/K/F
    Block          Type             File          Block
003055    1581.  Replaced      MSX    .SYS  000007       7.
003465    1845.  Replaced      DRV    .OBJ  000077      63.
037061   15921.  Replaced      < UNUSED >  010550    4456.
056106   23622.  Replaced      < UNUSED >  027575   12157.
056210   23688.  Replaced      < UNUSED >  027677   12223.
077521   32593.  Replaced      < UNUSED >  051210   21128.
143116   50766.  Replaced      < UNUSED >  043374   18172.
145337   51935.  Replaced      < UNUSED >  045615   19341.
?DUP-W-Bad blocks detected 8.
```

When you use /F with /I, you use it either to copy a file from an input device to an output device, or to copy an input device to an output file. Note that /I does not copy track 0 of diskettes. If you use a magtape for either the input or output device, you must specify a file name for the magtape followed by the /F option. Do not include wildcards in either the input or output file specification when you use the /F option.

6.3.5 Boot Option (/O)

The /O option can perform two operations: a hardware bootstrap of a specific device containing an RT–11 system, and a bootstrap of a particular RT–11 monitor file that does not affect the bootstrap blocks on the device.

The command syntax for a device bootstrap is as follows:

dev:/O

This operation has the same results as a hardware bootstrap. Valid devices for the boot operation follow:

DD0:-DD1:	DU0:-DU7:
DK:	DX0:-DX1:
DL0:-DL3:	DY0:-DY1:
DM0:-DM7:	RK0:-RK7:
DS0:-DS7:	SY:

NOTE

The following unsupported devices are also valid for the /O option:

DT0:-DT7:
DP0:-DP7:
PD0:-PD1:
RF:

Use the following syntax to boot a monitor without changing the bootstrap on the device:

dev:monitor-name/O

This makes it easy for you to switch from one monitor to another. Whether bootstrapping a specific monitor or a specific device, DUP checks to see if the bootstrap blocks are correctly formatted. If the boot operation you request is invalid, DUP prints an error message and waits for another command.

When you reboot with the /O option, you do not have to reenter the date and time of day with the monitor DATE and TIME commands. However, the clock does lose a few seconds during the reboot.

The following command reboots the RT–11 system under the SJ monitor:

```
*DL0:RT11SJ.SYS/O
RT-11SJ     V05.00
```

To boot a different monitor, for example the FB monitor (for DY0:), type:

```
*DY0:RT11FB.SYS/O
```

6.3.6 Boot Foreign Volume Option (/Q)

Use the /Q option with /O to boot a volume that has a monitor other than the RT–11 Version 4 or 5 monitor. Note that you must use /Q to boot any version of RT–11 previous to Version 4.

The following example boots an RT–11 Version 3B volume.

```
*DY0:/D/Q
RT-11SJ V03B-00B
```

DUP does not retain the date and time when you use the /Q option.

6.3.7 Squeeze Option (/S)

Use the /S option to compress a volume (disk, diskette) onto itself or onto another disk. To do this, DUP moves all the files to the beginning of the volume, producing a single, unused area after the group of files. The squeeze operation does not change the bootstrap blocks of a volume. Since it is critical to perform an error-free squeeze operation, be sure to scan a volume (with /K) before you use /S.

The output volume you specify, if any, must be an initialized volume. If you specify an output volume, DUP does not request confirmation before it performs the operation. If you do not specify an output volume, DUP prints the Are you sure? message and waits for your response before proceeding. You must type Y followed by a carriage return for the command to be executed.

The /S option does not operate on files with .BAD file types, preventing you from reusing bad blocks that occur on a disk. You can rename files containing bad blocks, giving them a .BAD file type, and therefore cause DUP to leave them in place when you execute a /S. During a squeeze operation, files with a .BAD file type are renamed FILE.BAD. DUP inserts files before and after .BAD files until the space between the last file it moved and the .BAD file is smaller than the next file to be moved.

If an error occurs during a squeeze operation, DUP continues the operation, performing it one block at a time. If no error message prints, you can assume that the operation completed correctly.

The syntax of the command is:

[output-device =]input-device/S

Do not use /S on the system device (SY:) when a foreground or system job is loaded. A ?DUP-F-Can't squeeze SY: while foreground loaded error message results if you attempt this, and DUP ignores the /S operation. You must unload the foreground job before using the /S option. Also, you should not attempt to squeeze any volume that a running foreground job is using. Data may be written over a file that the foreground job has open, thereby corrupting the file and possibly causing a system crash.

NOTE

If you perform a compress operation on the system volume, the system automatically reboots when the compress operation is completed. This occurs to prevent system crashes that can occur when a system file is moved.

You can use /X with /S to suppress the automatic reboot and leave DUP running. However, you should use /X only if you are certain that the monitor file will not move. Even then, you should reboot the system when the squeeze operation completes if the device handlers have moved.

The following examples use the /S option:

```
*SY:/S
SY:/Squeeze; Are you sure?Y

RT-11SJ     V05.00
```

The command shown above compresses the files on the system volume and reboots the system when the compress operation completes.

NOTE

> If you compress your system volume, make sure the DUP program has the name DUP.SAV. If not, a system failure may occur.

```
*DY0:*=DY1:/S
```

This command transfers all the files from device DY1: to device DY0:, leaving DY1: unchanged. The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

6.3.8 Extend Option (/T:n)

Use the /T option to extend the size of a file. The syntax of the command is:

filespec = /T:n

where:

filespec	represents the device, file name, and file type of the file to be extended
n	represents the number of blocks to add to the file

You can extend a file in this manner only if it is followed by an unused area at least n blocks long. Any blocks not required by the extend operation remain in the unused area.

The following example uses the /T option:

```
*DY1:ZYZ.TST=/T:100
```

This command assigns 100 more blocks to the file named ZYZ.TST on device DY1:.

6.3.9 Bootstrap Copy Option (/U[:xx])

In order to use a volume as a system volume, you must copy a bootstrap onto it. To do this, first make sure that the appropriate monitor file and handler are stored on the volume. For a double-density diskette system, for example, check to see that the file DY.SYS is in the diskette directory. If it is, then you can copy the desired monitor onto the diskette, using the /U option.

The option argument, xx, represents a target system device name. For example, you can use this argument when you are creating a bootable RX01 diskette if the current system is on an RX02 system.

NOTE

When you use the /U option, make sure that the input volume is also the output volume.

To copy a bootstrap for the SJ monitor on DL1:, for example, use the following procedure:

1. Obtain a formatted disk. (Most disks, diskettes, and DECtape II volumes are formatted by the manufacturer. However, Chapter 8, FORMAT, does outline the procedure for reformatting RK05, RK06, RK07, RP02, and RP03 disks, and RX01 and RX02 diskettes.)
2. Initialize the disk with /Z (see Section 6.3.13).
3. Copy files onto the disk.
4. Copy the monitor and RL02 handler, DL.SYS, onto the disk.
5. Copy the monitor bootstrap onto the disk with /U.

The following example shows how to initialize a diskette, copy files to it, and write a bootstrap onto the diskette:

```
*DY1:/Z/Y
```

The command shown above (step 2 of the procedure described above) initializes the diskette.

```
*DY1:*=DY0:/S
```

This command, which combines steps 3 and 4, squeezes all the files from DY0: onto DY1:.

```
*DY1:*=DY0:RT11FB.SYS/U
```

The last command (step 5) writes the bootstrap for the FB monitor onto the bootstrap blocks (blocks 0 and 2–5) of DY1:. The file name * is not significant; it is a dummy file name required by the Command String Interpreter.

6.3.10 Volume ID Option (/V[:ONL])

You can use the /V option as an action option to print the volume ID of a device or to change the volume ID.

The syntax of the command is:

device:[/Z]/V[:ONL]

where:

device: is the device whose volume ID you want to display or change

If you specify only /V, DUP prints out on the console terminal the volume ID and owner name of the device you specify. If you specify /Z with /V, DUP initializes the device and prompts you for a new volume ID and owner name. If you specify /Z/V:ONL, DUP assumes you want only to change the volume ID and owner name and not initialize the device.

When you specify either /Z/V or /Z/V:ONL, DUP prompts you for a volume ID:

```
Volume ID?
```

Respond with a volume ID that is up to 12 characters long for an RT–11 directory-structured volume or up to six characters long for magtape. Terminate your response with a carriage return. DUP then prompts for an owner name:

```
Owner?
```

Respond with an owner name that is up to 12 characters long for an RT–11 directory-structured volume or up to 10 characters long for magtape. Terminate your response with a carriage return. DUP ignores characters you type beyond the valid length.

You cannot change the volume ID of a magtape without initializing the entire tape. The /V:ONL command changes only the volume ID and owner name; it does not initialize the device. Section 6.3.13.2 describes how to use /V with the /Z option to initialize a device and write new volume identification on it.

The following example uses the /V:ONL option:

```
* DL1:/Z/V:ONL
DL0:/Volume ID change; Are you sure?  Y
Volume ID?  FORTRAN VOL
Owner?      Nancy
```

This command writes a new volume ID and owner name on device DL1:.

6.3.11 Wait for Volume Option (/W)

The /W option causes DUP to prompt you for the volumes to operate on, and waits for you to mount them. It is useful for single-disk systems or diskette systems. /W is a mode option that you can use with any of the action options, but you can specify only one action with it in a command line.

The /W option initiates execution of a command, but then pauses and prints the message Mount input volume in <device>; Continue?, where <device> represents the device into which you mount the input volume. At this time you can remove the system volume (if necessary) and mount the volume on which you actually want the operation to take place. When the new volume is loaded, type Y or any string beginning with Y followed by a carriage return to execute the operation. If you type N or any string beginning with N, or CTRL/C, the operation is not completed. Instead DUP prompts you to remount the system volume if you have removed it and returns control to the keyboard monitor. Any other response causes the message to repeat.

If you type Y, DUP prompts you for the input volume, if any. When the operation completes (except the /O operation, which boots the system), the Mount system volume in <device>; Continue? message prints. Replace the system device and type Y or any string beginning with Y followed by a carriage return. If you type any other response, DUP prompts you to mount the system volume until you type Y. When you type Y, the asterisk (*) prompt prints, and DUP waits for you to enter another command.

The following example uses the /W option:

```
*DY1:/K/F/W
Mount input volume in DY1; Continue? Y
?DUP-I-No bad blocks detected DY1:
Mount system volume in DY1; Continue? Y
*
```

This command directs DUP to scan the diskette for bad blocks. During the first pause, the system diskette is removed and another diskette is mounted. A Y is typed and the scan operation executes. During the second pause, the system disk (on which DUP is stored) is replaced and another Y is typed. DUP prompts for another command. When you use /W, make sure that DUP is on the system volume.

6.3.12 No Query Option (/Y)

Use the /Y option to suppress the query messages that some commands print.

Certain options normally print the Foreground job loaded, Continue? message if a foreground job is loaded when you issue one of them (/C, /I, /O, /Q, /S, /T, and /Z). You must respond to the query message by typing Y or any string beginning with Y followed by a carriage return for the operation to

proceed. Some other options (/C, /I, /O, /S, /V, and /Z) print the Are you sure? message and wait for your response. If a foreground job is loaded and you specify one of these options, DUP combines the two query messages into one message and waits for your response. You can suppress all these messages and the pause associated with them by specifying /Y in the command string.

Note, if you use /Y with /Z to initialize your system volume, the system ignores /Y.

6.3.13 Directory Initialization Option (/Z[:n])

You must initialize a device before you can store files on it. Use the /Z option to clear and initialize the directory of an RT–11 directory-structured device. The /Z option must always be the first operation you perform on a new device after you receive it, formatted, from a manufacturer. After you use /Z, there are no files in the directory.

The syntax of the command is as follows:

device:/Z[:n]

where:

device	represents the device you want to initialize.
:n	is an octal integer (greater than or equal to 1) that represents the size increase, in words, of each directory entry. DUP adds this number to the default number of words allocated for each entry (valid only for directory-structured devices).

The size of the directory determines the number of files that can be stored on a device. The system allows a maximum of 72 files per directory segment, and 31 directory segments per device. Each segment uses two blocks of disk space. If you do not specify n, each entry is seven words long (for file name, creation date, and file position information). When you allocate extra words, the number of entries per directory segment decreases. The formula for determining the number of entries per directory segment is:

(512–7)/((number of extra words) + 7)

For example, if you use /Z:1, you can make 63 entries per segment. RT–11 does not normally support nonstandard directory formats, and DIGITAL does not recommend altering the directory format.

6.3.13.1 Changing Directory Segments (/N:n) – If you do not want the default directory size of the device, use /N with /Z to set the desired number of directory segments for entries in the directory. The syntax is as follows:

/Z/N:n

In this option, n is an integer in the range 1–31 that represents the number of directory segments you want the directory to have.

Table 6–3 lists the default directory sizes, in segments, for RT–11-supported directory-structured devices.

Table 6–3: Default Directory Sizes

Device	Number (decimal) of Segments in Directory
DD	1
DL (RL01)	16
DL (RL02)	31
DM	31
DU (disk)	31
DU (diskette)	1
DX	1
DY (single-density)	1
DY (double-density)	4
RK	16

If the default directory size for diskettes is too small for your needs, see the *RT–11 Installation Guide* for details on increasing the default number of directory segments.

The following example initializes the directory on device DL1: and allocates six directory segments.

```
* DL1:/Z/N:6
DL1:/Initialize; Are you sure?  Y
```

6.3.13.2 Changing Volume ID (/V) – When you initialize a disk or magtape, DUP normally maintains the volume ID and owner name. If at initialization time you want to change the volume ID and owner name, use the /V option with /Z. For example, the following command initializes device DL1: and prompts you for a volume ID and owner name. Section 6.3.10 illustrates these prompts and shows how to use them.

```
* DL1:/Z/V
DL1:/Initialize; Are you sure?  Y
Volume ID?  VOUCHERS
Owner?  PAYABLES
```

6.3.13.3 Replacing Bad Blocks (/R[:RET]) – If you have RK06, RK07, RL01, or RL02 disks, you can use the /R option with either /I or /Z.

Use this option with /I to preserve the output volume's bad block replacement table. DUP copies all blocks from the input volume to the output volume except those blocks that contain the input volume's bad block replacement table.

Use this option with /Z to scan a disk for bad blocks. If DUP finds any bad blocks, it creates a replacement table so that routine operations access good blocks instead of bad ones. Thus, the disk appears to have only good blocks.

Note, though, that accessing this replacement table slows response time for routine input and output operations. If you use :RET with /R, DUP initializes the volume and retains the bad block replacement table (and FILE.BAD files) created by the previous /R command. Note that the :RET argument is invalid when you use /R with /I.

Note that the monitor file cannot reside on a block that contains a bad sector error (BSE) if you are doing bad block replacement. If this condition occurs, a boot error results when you attempt to bootstrap the system. If this occurs, move the monitor.

With an RK06, RK07, RL01, or RL02 you have the option of deciding which bad blocks you want replaced if the number of bad blocks exceeds what can fit in the replacement table (replacement table overflow). The RK06s and RK07s support up to 32(decimal) bad blocks in the replacement table; the RL01s and RL02s support up to 10.

With an RK06 or RK07 disk, DUP can replace only those bad blocks that generate a BSE. Of the blocks DUP cannot replace, DUP can report a bad block as being hard or soft. If you perform two bad block scans and a block is reported as bad in both reports, this indicates a hard error. If in the second report the block is not reported as bad, the block has recovered from a soft error.

With an RL01 or RL02, DUP can replace any kind of bad block. The following paragraphs describe how to designate which blocks to replace on an RK06, RK07, RL01, or RL02 disk.

When you use /R, DUP prints out a list of replaceable bad blocks as in the following sample:

```
    Block          Type
030722  12754.  Replaceable
115046  39462.  Replaceable
133617  46991.  Replaceable
136175  48253.  Replaceable
136277  48319.  Replaceable
136401  48385.  Replaceable
140405  49413.  Replaceable
146252  52394.  Replaceable
?DUP-W-Bad blocks detected 8.
```

If there is a replacement table overflow, DUP prompts you to indicate which blocks you want replaced as follows:

```
?DUP-W-Replacement table overflow DEV:
Type <RET>, 0, or nnnnnn (<RET>)
Replace block #
```

The value nnnnnn represents the octal block number of the block you want the system to replace.

After you enter a block number, DUP responds by repeating the Replace block # prompt. Type a 0 at any time if you do not want any more blocks replaced, and this will end prompting. DUP marks any blocks not placed in the replacement table as FILE.BAD.

If you enter a carriage return at any time, DUP places all bad blocks you have not entered into the replacement table, starting with the first on the disk, until the table is full. DUP assigns the name FILE.BAD to any remaining bad blocks and prompting ends.

If you use /Y with /R, the effect will be as if you entered a carriage return in response to the first Replace block # prompt.

6.3.13.4 Covering Bad Blocks (/B[:RET]) – To scan the volume for bad blocks and write files over them, use the /B option with /Z. For every bad block DUP encounters on the device, it creates a file called FILE.BAD to cover it. After the disk is initialized and the scan completed, the directory consists only of FILE.BAD entries that cover the bad blocks. If DUP finds a bad block in the boot block or the directory, it prints an error message and the disk is not usable.

If you specify :RET with /B, DUP will retain through initialization all FILE.BAD files created by a previous /B. If you use the /B option to initialize a volume that has been previously initialized using the /R option, DUP creates FILE.BAD files to cover the bad blocks on the volume, and the system then ignores the bad block replacement table.

If the volume being initialized contains bad blocks, the system prints the locations of the bad blocks in octal and in decimal, as in the following example:

```
* DY0:/Z/B
DY0:/Initialize: Are you sure ?  Y
     Block          Type
000120      80.  Hard
000471     313.  Hard
000521     337.  Hard
?DUP-W-Bad blocks detected 3.
```

The left column lists the locations in octal, and the middle column lists the locations in decimal. The right column indicates the type of bad block found: hard or soft.

6.3.13.5 Restoring a Disk (/D) – Use /D to uninitialize (restore) a volume if you have not transferred any files to it since initialization. DUP will restore all files and directory entries that were present before the volume was initialized. This option is useful if you initialize a volume by mistake. However, you cannot restore volumes that support bad block replacement if bad blocks were found during initialization.

The following command restores volume DY1:.

```
* DY1:/Z/D
```

Note that /D does not restore boot blocks. Thus, if you use /D to restore a previously bootable volume, use the bootstrap copy option, /U[:xx], to make the volume bootable again.

Chapter 7
File Exchange Program (FILEX)

The file exchange program (FILEX) is a general file transfer program that converts files from one format to another so that you can use them with various operating systems. You can copy files between any block-replaceable RT–11 directory-structured device and any device listed in Table 7–1.

Table 7–1: Supported FILEX Devices

Device	Valid as Input	Valid as Output
PDP–11 DOS/BATCH DECtape	X	X
DOS/BATCH disk	X	
RSTS DECtape	X	X
DECsystem–10 DECtape	X	
Interchange diskette (RX01, RX02 single-density, PDT–11/150)	X	X

FILEX does not support magtapes, cassettes, or double-density diskettes in any operation. Note that you can transfer only one file at a time to interchange diskette format.

Section 4.2 of the *RT–11 System User's Guide* describes how to use wildcards, which you can use in the FILEX command string. The default device for all FILEX operations is DK:. You can use wildcards when transferring

from interchange to RT–11 format. However, you cannot use embedded wildcards in any file name or file type. For example, the following line represents a valid file specification.

```
*
  *.MAC
```

The next line is an invalid file specification for FILEX.

```
*
  T%ST.MAC
```

7.1 File Formats

FILEX can transfer files created by four different operating systems: RT–11, DECsystem–10, universal interchange format (IBM) system (see the *RT–11 Software Support Manual*), and DOS/BATCH (PDP–11 Disk Operating System). You can use the following three data formats in a transfer: ASCII, image, and packed image. ASCII files conform to the American Standard Code for Information Interchange in which each character is represented by a 7-bit code. In ASCII mode, FILEX deletes null and rubout characters, as well as parity bits.

NOTE

If you attempt to use RT–11 volumes for both input and output, FILEX generates an error message.

Because the file structure and data formats for each system vary, options are needed in the command line to indicate the file structures and the data formats involved in the transfer. These options are discussed in Section 7.3. FILEX assumes that all devices are RT–11-structured. You can use options to indicate otherwise.

7.2 Calling and Terminating FILEX

To call FILEX from the system device, respond to the keyboard monitor prompt by typing:

```
.
  R FILEX (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter a command.

Type CTRL/C to halt FILEX when it is waiting for console terminal input and return control to the monitor. To restart FILEX, type R FILEX or REENTER in response to the monitor's dot.

7.3 Options

Table 7–2 lists the options that initiate various FILEX operations. The table contains four categories: transfer, operation, modifier, and device.

Transfer options direct FILEX to copy data in a certain mode. The three transfer modes are: ASCII, image, and packed image.

Operation options perform other functions in addition to the data transfer. These additional functions include deleting files, producing listings, and zeroing device directories. FILEX accepts one transfer option and one operation option in a single command.

Modifier options cause transfers and operations to be performed in a certain manner. For example, when you use the /Y option to modify the /Z option, FILEX suppresses the /Init are you sure? message. There are three modifier options: /V[:ONL], /W and /Y.

Device options indicate the formats of devices that are involved in a transfer. These formats are DOS/BATCH or RSTS, DECsystem–10, and interchange. You can specify one device option for each file involved in the transfer. The device options (/S, /T, and /U) must appear following the device and file name to which they apply; other options may appear anywhere in the command line. These options are explained in more detail in the sections following Table 7–2.

Table 7–2: FILEX Options

Options	Function
Transfer	
/A	Indicates a character-by-character ASCII transfer in which FILEX deletes rubouts and nulls. If you use /U with /A, FILEX also ignores all sector boundaries on the diskette and assumes that records are to be terminated by a line feed, vertical tab, or form feed. If you use /A with /T, FILEX assumes that each PDP–10 36-bit word contains five 7-bit ASCII bytes. The transfer terminates when a CTRL/Z is encountered. (This feature is included for compatibility with RSTS.) FILEX does not transfer the CTRL/Z.
/I	Performs an image mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer is word-for-word. If the input is from DECsystem–10, /I indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /I option. In this case, each PDP–10 36-bit word will contain one PDP–11 8-bit byte in its low-order bits. If input or output is an interchange diskette, FILEX reads and writes four diskette sectors for each RT–11 block.
/P	Performs a packed image mode transfer. If the input is DOS/BATCH, RSTS, or RT–11, the transfer will be word-for-word. If the input is from DECsystem–10, /P indicates that the file resembles a file created on DECsystem–10 by MACY11, MACX11, or LNKX11 with the /P option. In this case, each PDP–10 36-bit word will contain four PDP–11 8-bit bytes aligned on bits 0, 8, 18, and 26. This is the default mode. If the output is interchange diskette, FILEX writes the data as EBCDIC.

(Continued on next page)

Table 7–2: FILEX Options (Cont.)

Options	Function
Operation	
/D	Deletes the file you specify from the device directory. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
/F	Produces a brief listing of the device directory on the terminal. It lists only file names and file types. FILEX can only list directories of block-replaceable devices, and those directories only on the console terminal.
/L	Produces a complete listing of the device directory on the console terminal, including file names, block lengths, and creation dates.
/Z	Initializes the directory of the device you specify. This option is valid only for DOS/BATCH, RSTS DECtape, and interchange diskette.
Modifier	
/V[:ONL]	/V is used with /Z and /U[:n] together to write a volume identification on an interchange diskette during initialization. A volume identification can be up to six characters long. Using /V:ONL with /Z and /U[:n] changes only the ID and does not initialize the interchange diskette. You can also use /V[:ONL] with /F or /L to list the volume identification of an interchange diskette as well as its directory.
/W	Transfers files in a single- or small-disk system. FILEX initiates the transfer, but pauses and waits for you to mount the volumes involved in the transfer.
/Y	Used with /Z to suppress the dev:/Init are you sure? message.
Device	
/S	Indicates that the device is a valid DOS/BATCH or RSTS block-replaceable device.
1/T	Indicates that the device is a valid DECsystem–10 DECtape.
/U[:n.]	Indicates that the device is an interchange diskette. The symbol n. represents the length of each output record, in characters. The argument n. is a decimal integer in the range 1–128. The default value is 80; n. is not valid with an input file specification, or with /A or /I.

7.3.1 Transferring Files Between RT–11 and DOS/BATCH or RSTS (/S)

You can transfer files between block-replaceable devices used by RT–11 and the PDP–11 DOS/BATCH system. Input from DOS/BATCH may be either disk or DECtape. You can use both linked and contiguous files.

If the input device is a DOS/BATCH disk, you should specify a DOS/BATCH user identification code (UIC) in the form [nnn,nnn]. The initial default value is [1,1]. The UIC you supply will be the default for all future transfers.

If you do not specify a UIC, FILEX will use the current default UIC. Note that the square brackets ([ ]) are part of the UIC; you must type them when you specify a UIC.

Output to DOS/BATCH is limited to DECtape only. You do not need a UIC in a command line where you are accessing only DECtape. Individual users do not own files on DECtape under DOS. However, no error occurs if you do use a UIC. DECtape used under the RSTS system is valid as both input and output, since its format is identical to DOS/BATCH DECtape. You may use any valid RT–11 file storage device for either input or output in the transfer. The RT–11 device DK: is assumed if you do not indicate a device.

An RT–11 DECtape can hold more information than a DOS/BATCH or RSTS DECtape. When you copy files from a full RT–11 tape to a DOS DECtape, some information may not transfer. In this case, an error message prints and the transfer does not complete.

When a transfer from an RT–11 device to a DOS DECtape occurs, the block size of the file can increase. However, if the file is later transferred back to an RT–11 device, the block size does not decrease.

To transfer a file from a DOS/BATCH block-replaceable device or RSTS DECtape to an RT–11 device, type a command with the following syntax:

output-filespec = input-filespec/S[/option]

where:

output-filespec	represents any valid RT–11 device, file name, and file type (if the device is not file structured, you may omit the file name and file type).
input-filespec	represents the DOS/BATCH or RSTS device, UIC, file name, and file type to be transferred. (See Table 7–1 for a list of valid devices.)
/S	is the option that designates a DOS/BATCH or RSTS block-replaceable device. (This option must be included in the command line.)
/option	is one of the three transfer options from Table 7–2, and the /W modifier option if necessary.

To transfer files from an RT–11 storage device to a DOS/BATCH or RSTS DECtape, type a command with the following syntax:

DTn:output-filename/S[/option] = input-filespec

where:

DTn:output-filename	represents the file name and file type of the file to be created, as well as the DOS/BATCH or RSTS DECtape on which to store the file.
input-filespec	represents the device, file name, and file type of the RT–11 file to be transferred.

/S	is the option that designates a DOS/BATCH or RSTS DECtape. (This option must be included in the command line.)
/option	is one of the three transfer options from Table 7–2, and the /W modifier option if necessary.

The following examples illustrate the use of the /S option.

The following command instructs FILEX to transfer a file called SORT.ABC from the RT–11 default device DK: to a DECtape in DOS/BATCH or RSTS format on unit DT2. The transfer is in image mode.

```
*DT2:SORT.ABC/S=SORT.ABC/I
```

The next command allows a file to be transferred from DOS/BATCH (or RSTS) DECtape to the line printer under RT–11. The transfer is done in ASCII mode.

```
*LP:=DT2:FIL.TYP/S/A
```

The next command causes the file MACR1.MAC to be transferred from the DOS/BATCH disk on unit 1, stored under the UIC [1,2], to the RT–11 device DK:. [1,2] becomes the default UIC for any further DOS/BATCH operations.

```
*DK:*.*=RK1:[1,2]MACR1.MAC/S
```

7.3.2 Transferring Files Between RT–11 and Interchange Diskette (/U)

You can transfer files between block-replaceable devices used by RT–11 and interchange format diskettes. Files are transferred in one of three formats: ASCII, image, and packed image EBCDIC mode.

A universal diskette consists of 77 tracks (some of which are reserved), each containing 26 sectors numbered from 1 to 26. A sector contains one record of 128 or fewer characters. When an interchange diskette is in packed image mode, records always begin on a sector boundary. There is only one record per sector. If a record does not fill a sector, the remainder is filled with blanks. Since packed image EBCDIC mode is inefficient and wastes space, packed image mode is recommended only to read or write diskettes that must be compatible with IBM 3741 format. Packed image (EBCDIC) mode is generally compatible with IBM 3741 format. (Although IBM 3741 format supports error mapping of bad sectors and multivolume files, FILEX does not.) Packed image (EBCDIC) is the default mode, so you must use one of the options from Table 7–2 to specify ASCII or image mode. All records of a file must be the same size. You indicate this with the /U:n. option.

NOTE

> File types are not usually recognized in interchange format; instead, a single, 8-character file name is used. However, in order to provide uniformity throughout RT–11, FILEX has

been designed to accept a 6-character file name with a 2-character file type. If you transfer a file from RT–11 to interchange diskette, any 3-character file type is truncated to two characters.

To transfer files from RT–11 format to interchange format, type a command with the following syntax:

output-filespec/U[:n.][/option] = input-filespec

where:

output-filespec	represents the device, file name, and file type of the interchange file to be created. Note that you cannot use wildcards in the output file specification.
/U[:n.]	is the option that designates an interchange diskette. This option must be included in the command line. The argument n. represents the length of each output record, in characters; n is a decimal integer in the range 1 to 128 (default is 80). The argument n is invalid with either /A or /I.
/option	is one of the three transfer options from Table 7–2, and the /W modifier option if necessary.
input-filespec	represents the device, file name, and file type of the RT–11 file to be transferred. The file name is six characters long, with a 2-character file type. Any 3-character file type is truncated to two characters.

To transfer files from interchange diskette to RT–11 format, type a command with the following syntax:

output-filespec = input-filespec/U[/option]

where:

output-filespec	represents the device, file name, and file type of the RT–11 file to be created. Note that you can use wildcards as input.
input-filespec	represents the device, file name, and file type of the interchange file to be transferred.
/U	is the option that designates an interchange diskette. (This option must be included in the command line.)
/option	is one of the three transfer options from Table 7–2, and the /W modifier option if necessary.

The following command transfers the file IVAN.CAT from RT–11 RK05 unit 2 to the diskette on unit 1. The transfer is done in exact image mode (indicated by /I), ignoring all sector boundaries.

```
* DX1:IVAN.CA/U/I=RK2:IVAN.CAT
```

The next command instructs FILEX to transfer the file BENMAR.FRM from the RT–11 disk unit 2 to the diskette on unit 0, and rename it KENJOS.JO. The /U option indicates that the format is to be changed from ASCII to the interchange format. There will be one record per sector of 128 or fewer characters. If there are fewer than 128 characters, the remainder of the sector will be filled with spaces.

```
* DX0:KENJOS.JO/U=RK2:BENMAR.FRM
```

The next command transfers the file TYPE.SET from RT–11 diskette unit 0 to the interchange diskette on unit 2. The exchange converts ASCII to interchange format, putting a maximum of seven (indicated by :7.) characters into each sector until the entire record has been transferred. Records in excess of seven characters will be broken up and placed in succeeding sectors on the diskette. New records always begin on a sector boundary; carriage returns and line feeds are discarded. However, if you use /A or /I, FILEX ignores boundary limits and preserves carriage returns and line feeds.

```
* DX2:TYPE.SE/U:7.=DX0:TYPE.SET
```

File TYPE.SET before transfer:

ABCDEFGHIJKLMN

File TYPE.SET after transfer:

ABCDEFG----(spaces up to 128 characters) Sector 1
HIJKLMN----(spaces up to 128 characters) Sector 2

The next command copies file IVAN.CA from the interchange diskette on unit 1 to the RT–11 line printer, treating the input as ASCII characters. Note that once a record has been divided into sectors, it cannot be transferred back to its original size.

```
* LP:=DX1:IVAN.CA/U/A
```

7.3.3 Transferring Files to RT–11 from DECsystem–10 (/T)

Output may be to any valid RT–11 device. DECsystem–10 DECtape is the only valid input device.

To transfer files from DECsystem–10 format to RT–11 format, use this command syntax:

output-filespec = input-filespec/T[/option]

where:

output-filespec	represents any valid RT–11 device, file name, and file type. (If the device is not file-structured, you can omit the file name and file type.)
input-filespec	represents the DECtape unit, file name, and file type of the DECsystem–10 file to be transferred.

/T — is the option that signifies a DECsystem–10 DECtape (When you use /T, and especially when you also use /A, the system clock loses time. Examine the time, and reset it if necessary with the TIME command.)

/option — is one of the three transfer options from Table 7–2, and the /W modifier option if necessary.

You cannot convert RT–11 files to DECsystem–10 format directly. However, there is a two-step procedure for doing this. First, run RT–11 FILEX and convert the files to DOS formatted DECtape. Then run DECsystem–10 FILEX to read the DOS DECtape.

The following command converts the ASCII file STAND.LIS from DECsystem–10 ASCII format to RT–11 ASCII format and stores the file under RT–11 on DECtape unit 2 as STAND.LIS.

```
*DT2:STAND.LIS=DT1:STAND.LIS/T/A
```

Transfers from DECsystem–10 DECtape to RT–11 may cause an <UNUSED> block to appear after the file on the RT–11 device. This is a result of the way RT–11 handles the increased amount of information on a DECsystem–10 DECtape.

The next command indicates that all files on the DECsystem–10 formatted DECtape on unit 0 with the file type .LIS are to be transferred to the RT–11 system device using the same file name and a file type of .NEW. The /P option is the assumed transfer mode.

```
*SY:*.NEW=DT0:*.LIS/T
```

Files may not be transferred to RT–11 devices from a DECsystem–10 DECtape if a foreground job is running. This restriction is due to the fact that when FILEX reads DECsystem–10 files, it accesses the DECtape control registers directly instead of using the RT–11 DECtape handler.

7.3.4 Listing Directories (/L)

You can list at the terminal a directory of any of the block-replaceable devices used in a FILEX transfer. The command syntax is:

device:/L/option

where:

device — represents the block-replaceable device. These are the valid device types:

DOS/BATCH, RSTS DTn:, RKn:

DECsystem–10 DTn:

Interchange diskette DXn: DYn:

/L is the listing option. (You can substitute /F if you want a brief listing of file names only.)

/option is /S, /T, or /U, and the /W modifier option if necessary. These are the valid format and option combinations:

DOS/BATCH, RSTS /S

DECsystem–10 /T

Interchange diskette /U

The following example shows the complete disk directory for UIC[1,7] of the device RK1:. The letter C following the file size on a DOS/BATCH or RSTS directory listing indicates that the file is contiguous.

```
* RK1:/L/S
 18-FEB-83
BADB      .SYS     1       18-FEB-83
MONLIB    .CIL   175C      18-FEB-83
DU11      .PAL    45       18-FEB-83
VERIFY    .LDA    67C      18-FEB-83
CILUS     .LDA    39       18-FEB-83
```

The next example is a command that lists all files with the file type .PAL that are stored on DECtape unit 1.

```
* DT1:*.PAL/L/S
```

The next command produces a brief directory listing of the interchange diskette on unit 0, giving file names only.

```
* DX0:/U/F
```

The following command lists all files on the DECsystem–10 formatted DECtape on unit 1, regardless of file name or file type; with the /F, a brief directory is requested in which only file names print.

```
* DT1:*.*/F/T
```

7.3.5 Deleting Files from DOS/BATCH (RSTS) DECtapes and Interchange Diskettes (/D)

Use FILEX to delete files from DOS/BATCH and RSTS formatted DECtapes, and from interchange diskettes.

To delete files, type a command with the following syntax:

filespec/D/option

where:

filespec represents the device, file name, and file type of the file to be deleted.

/D is the delete option.

/option can be either /S, for DOS/BATCH and RSTS block-replaceable devices, or /U, for interchange diskettes. You can also include the /W modifier option, if necessary.

The following command deletes all files with the file type .PAL on DECtape unit 0.

```
* DT0:*.PAL/D/S
```

The next command deletes the file TABLE.OBJ from the DECtape on unit 2.

```
* DT2:TABLE.OBJ/D/S
```

The next command deletes all files with an .RNO file type from the interchange diskette on unit 0.

```
* DX0:*.RN/D/U
```

7.3.6 Initializing the Directories of DECtapes and Interchange Diskettes (/Z)

You can also use FILEX to initialize the directories of DOS/BATCH DECtapes, RSTS DECtapes, and interchange diskettes.

Use this command syntax:

device:/Z/option[/Y]

where:

device represents the DOS/BATCH or RSTS DECtape, or the interchange diskette to, be zeroed.

/Z is the initialize option.

/option can be either /S, for DOS/BATCH and RSTS DECtapes, or /U, for interchange diskettes. You can also include the /W modifier option, if necessary.

/Y inhibits the FILEX confirmation message.

The following command directs FILEX to initialize the directory of the interchange diskette on unit 0.

```
* DX0:/Z/U
```

FILEX prints a confirmation message:

```
DX0:/Initialize; are you sure?
```

Respond with a Y or any string beginning with Y followed by a carriage return for initialization to begin. Any other response aborts the command.

The next command initializes the DECtape on unit 1 in DOS/BATCH (RSTS) format. Note that by using the /Y option you suppress the confirmation message.

```
* DT1:/Z/S/Y
```

NOTE

The directory of an initialized interchange diskette has a single file entry, DATA, that reserves the entire diskette. You must delete this file before you can write any new files on the diskette. This is necessary for IBM compatibility. Do this by using the following command:

```
* DX0:DATA/D/U
```

7.3.7 Interchange Diskette Volume ID Option (/V[:ONL])

The /V option enables you to write a volume identification on an interchange diskette when it is initialized. This option is used with the /U[:n] and /Z options together. You can also use /V[:ONL] with /L or /F to list a volume ID.

When you use this option, FILEX prompts you for a volume ID. Respond by typing a volume identification of up to six characters. Any string over six characters is truncated. If you type only a carriage return in response to the volume ID prompt, the default volume ID RT11A is written on the interchange diskette.

Use /V:ONL to change only the volume ID without initializing the interchange diskette.

The following command initializes an interchange diskette and writes a volume identification:

```
* DX0:/Z/U/V
 Volume ID?  Nancy
```

The next command changes only the volume ID of an interchange diskette.

```
* DX0:/Z/U/V:ONL
 Volume ID change; are you sure?  Y
 Volume ID?  Nancy
```

7.3.8 Wait Option (/W)

The /W option permits you to replace the system volume with another volume during an operation. You can use the /W option for a delete, directory listing, and initialization operation on a single-disk system, or to copy files between volumes when the system volume is neither the input nor the output volume if you have two drives available. When you use the /W option, you cannot use wildcards in the input specification.

When you use the /W option, FILEX guides you through a series of steps in the process of completing the operation. After you enter the initial command string, FILEX prints a message telling you which volume to mount. After you complete each step, type Y or any string beginning with Y followed by a carriage return to proceed to the next step. If you type N or any string beginning with N, or CTRL/C, FILEX prompts you to mount the system volume if you have removed it and the operation is not performed. Any other response causes the message to repeat.

When the operation is complete, FILEX prints a message instructing you to mount your system volume. Mount the system volume and type Y or any string beginning with Y followed by a carriage return. If you type any other response, FILEX prompts you to mount the system volume until you type Y.

When you use /W, make sure that FILEX is on your system volume.

The procedure for copying files with /W follows:

With your system volume mounted, enter a command string according to the FILEX syntax. After you have entered the command string, FILEX responds with the message:

```
Mount input volume in <device>; Continue?
```

Type Y or any string beginning with Y followed by a carriage return to continue the operation when you have mounted the input volume. FILEX then prints:

```
Mount output volume in <device>; Continue?
```

Type Y or any string beginning with Y followed by a carriage return to continue the operation after you have mounted the output volume.

When the file transfer is complete, FILEX prints the following message if you had to remove the system volume from <device>:

```
Mount system volume in <device>; Continue?
```

Type Y or any string beginning with Y followed by a carriage return to terminate the copy operation. If you type any other response, FILEX prompts you to mount the system volume until you type Y.

Chapter 8
Volume Formatting Program (FORMAT)

The FORMAT utility program formats disks and diskettes. You can also use FORMAT to convert single-density diskettes to double-density and vice versa. FORMAT can format RX01/RX02 diskettes, RK05 disks, and RK06/RK07 disks.

Formatting a volume makes that volume usable by RT–11. When you format a volume, FORMAT writes headers on each block in that volume. The header of a block contains data the device controller uses to transfer information to and from that block.

When you use FORMAT to convert a single-density diskette to double density, or vice versa, FORMAT writes media density marks on each block of the diskette. You can format a diskette only in a double-density diskette drive, DY:. If you attempt to format a diskette in a single-density diskette drive, DX:, FORMAT prints an error message.

Reformatting with the FORMAT program can also eliminate bad blocks that disks and diskettes sometimes develop as a result of age and use. Although formatting does not guarantee that each bad block will be eliminated, formatting can reduce the number of bad blocks.

NOTE

FORMAT destroys any data that currently exists on the disk.

8.1 Calling and Terminating FORMAT

To call FORMAT from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
R FORMAT (RET)
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, FORMAT prints its current version number. You can type CTRL/C to halt FORMAT and return control to the monitor when FORMAT is waiting for input from the console terminal. You cannot halt FORMAT during an operation by typing two CTRL/Cs.

If you interrupt the program during a formatting operation by some other means, the disk or diskette involved is not completely formatted. You must restart the operation on the same disk or diskette and allow it to run to completion.

8.2 FORMAT Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line that system utility programs accept. FORMAT accepts one device specification (either a physical or logical device name) followed, if necessary, by one or more options. An RK05 disk you wish to format can be located in any unit (0–7) of device RK:. A diskette you need to format must be mounted on an RX02 device (device DY:), but it can be located in any unit (0–3) of that device. You cannot format diskettes on an RX01 device.

8.3 FORMAT Confirmation Prompts

FORMAT automatically prints the device-name:/Are you sure? message before it begins any operation. The device name that prints out in the message is the physical name of the device you specify in the command line. Therefore, if you use a logical device name in the command line, the device name that FORMAT displays in the confirmation message is different from the name you type. If you want the operation to continue, type Y or any string beginning with Y followed by a carriage return in response to the confirmation message. Type N or CTRL/C to prevent the formatting operation from occurring. Any other response causes FORMAT to repeat the prompt.

You can use FORMAT from an indirect command file. To satisfy the device-name:/Are you sure? message, enter a Y as the next line of the indirect file immediately following the FORMAT command line. You can suppress the confirmation message completely by using the /Y option in the FORMAT command line. If you use /Y, you do not need to enter the Y on the following line.

If you try to format a volume while a foreground job is loaded, the system prints the following message.

```
Foreground Loaded.
<dev:>/FORMAT-Are you sure?
```

Type Y or any string beginning with Y followed by a carriage return to continue with the formatting operation. Type N or any string beginning with N, or CTRL/C, to abort the operation. Any other response causes the message to repeat.

NOTE

Although you can format or verify a volume while a foreground job is loaded, it is not recommended. If you try to format or verify a volume that the foreground job is using, data

on the volume will be written over and corrupted, which may cause the foreground job or the system to crash.

If you try to format a volume that contains protected files, the system prints the following message.

```
Volume contains protected files;  Are you sure?
```

Type Y or any string beginning with Y to continue the formatting operation. Type N or any string beginning with N, or CTRL/C, to abort the operation. Any other response causes the message to repeat.

After you format a disk, you should use the INITIALIZE command to prepare the volume for use with RT–11. See Chapter 4 of the *RT–11 System User's Guide* for more information on the INITIALIZE command.

8.4 Options

Options that you specify in a command line to the FORMAT program perform several functions. Table 8–1 summarizes these options and the operations they perform. You can combine these options, if necessary, in any order. More detailed explanations of the options are arranged alphabetically by option name in the sections that follow the table.

Table 8–1: FORMAT Options

Option	Section	Function
none	8.4.1	If you do not supply an option, FORMAT formats the volume you specify. If you specify an RX01 or RX02 diskette, the default operation that occurs is double-density diskette formatting. You can use /Y and /W with the default operation.
/P:n	8.4.2	Pattern verification option, where n represents an octal integer in the range 0 to 177777. The option specifies the specific 16-bit word pattern that FORMAT uses to write to the volume, and read from the volume, during the process of verification. If you do not use this option, FORMAT defaults to /P:200.
/S	8.4.3	Single-density option. This option formats a diskette in a single-density format.
/V[:ONL]	8.4.4	Verification option. When you specify /V in the command line, FORMAT first formats the specified volume, then verifies it. If you specify /V:ONL, FORMAT only verifies the specified device. Note that you can use /V:ONL with any disk, diskette, or DECtape II.
/W	8.4.5	Wait option. This option permits you to substitute another volume for the volume you specify in the command line, format the second volume, then replace the original volume. Note that this option is invalid for RC25 disks, RD51 disks, and RX50 diskettes.
/Y	8.4.6	No query option. This option suppresses the Are you sure? message FORMAT automatically prints before each operation.

8.4.1 Default Format

To format diskettes in double-density mode, specify the device name in the command line. You can also use /Y to suppress the query message, and /W to pause for a volume substitution. The following example formats the diskette in DY: evice unit 1 as a double-density diskette.

```
*DY1:
DY1:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
*
```

To format an RK05 or RK06/07 disk, specify the device name in the command line. You can also use /Y to suppress the query message and /W to pause for a volume substitution. The following example formats an RK05 disk in RK: device unit 1:

```
*RK1:
RK1:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
*
```

When you format an RK06 or RK07 disk, FORMAT lists the block numbers of all the bad blocks in the manufacturer's bad block table and in the sofware bad block table.

8.4.2 Pattern Verification Option (/P:n)

When you use the /P:n option with /V[:ONL] in the command line, you can specify the 16-bit word pattern you want FORMAT to use when it performs volume verification. The argument n represents an octal integer in the range 0 to 177777 that specifies the pattern or successive patterns you want FORMAT to use. Table 8–2 lists the verification patterns FORMAT uses and the corresponding values of n.

In /P:n, the number you specify for n indicates the value for the bit patterns to be run during verification. Bits set in /P:n select the patterns to be run. Table 8–2 shows which bit, when set, corresponds to each 16-bit verification pattern. To calculate the equivalent value of n, convert the bit set to an octal number. For example, FORMAT runs pattern 3 when bit 2 is set. When bit 2 alone is set, the equivalent octal number is 4.

Table 8–2 gives the equivalent n value for each verification bit pattern. If you want to run more than one bit pattern, add the values of n for the patterns you select. For example, suppose you want to run bit patterns 1, 3, and 5. The corresponding values of n are 1, 4, and 20, for a sum of 25. This is the value of n you would specify with /P to run all three bit patterns.

FORMAT converts the number you specify into a binary number; the number of each set bit specifies which patterns to run. The number 25 translates to the binary number 010 101. In the number 010 101, bits 0, 2, and 4 are

Table 8–2: Verification Bit Patterns

Pattern	Bit Set	n	16-Bit Pattern
1	0	1	000000
2	1	2	177777
3	2	4	163126
4	3	10	125252
5	4	20	052525
6	5	40	007417
7	6	100	021042
8	7	200	104210
9	8	400	155555
10	9	1000	145454
11	10	2000	146314
12	11	4000	*
13	12	10000	*
14	13	20000	*
15	14	40000	*
16	15	100000	*

* These patterns are reserved for future use. Currently these bit patterns run the default bit pattern (pattern 8).

set. As Table 8–2 shows, bit 0 specifies pattern 1, bit 2 specifies pattern 3, and bit 4 specifies pattern 5. If you specify /P:25, FORMAT runs patterns 1, 3, and 5. If you specify /P:255, FORMAT runs patterns 1, 3, 4, 6, and 8. If you specify /P:777, FORMAT runs patterns 1 though 9 during verification. If you do not use the /P:n option, FORMAT runs only pattern 8.

When you use the /P:n option, and you specify more than one pattern, FORMAT runs each pattern successively. After it completes verification, FORMAT prints at the terminal each bad block it found during each verification pass. The format of the verification report is:

```
PATTERN #x
----------
nnnnnn
.
```

In the example above, x represents the pattern number, and nnnnnn represents the bad block number. FORMAT makes a separate verification pass for each pattern it runs, and reports on each pass.

The sample command line that follows formats volume RK1: and verifies it with patterns 4, 5, and 6.

```
*RK1:/V/P:70
RK1:/FORMAT-Are you sure? Y
?FORMAT-I-Formatting complete
PATTERN #6
PATTERN #5
PATTERN #4
?FORMAT-I-Verification complete
*
```

The next sample command line verifies volume DL0: with pattern 2.

```
* DL0:/V:ONL/P:2
DL0:/VERIFY-Are you sure?   Y
PATTERN #2
?FORMAT-I-Verification complete
```

8.4.3 Single-Density Option (/S)

Use /S to format a diskette in single-density mode. You can also use /Y to suppress the query message and /W to pause for a volume substitution.

The following example formats the diskette in DY: device unit 1 as a single-density diskette.

```
* DY1:/S
DY1:/FORMAT-Are you sure?   Y
?FORMAT-I-Formatting complete
*
```

8.4.4 Verification Option (/V[:ONL])

Use the /V[:ONL] option to provide a verification of all blocks on a volume immediately following formatting. If you use the optional argument :ONL, FORMAT executes only the verification procedure. Although FORMAT can format only a limited assortment of storage volumes, it can verify any disk, diskette, or DECtape II.

In the process of verifying a storage volume, FORMAT first writes a 16-bit word pattern on each block of the specified volume, and then reads each pattern. For each read or write error it encounters, FORMAT prints at the terminal the block number for each block that generated the error.

NOTE

FORMAT destroys data on any storage volume it verifies.

The following command line uses /V to verify an RK05 disk after formatting.

```
* RK0:/V
RX0:/FORMAT-Are you sure?   Y
?FORMAT-I-Formatting complete
PATTERN #8
?FORMAT-I-Verification complete
```

The next example uses /V:ONL to verify, but not format, an RX01.

```
* DX1:/V:ONL
DX1:/VERIFY-Are you sure?   Y
PATTERN #8
?FORMAT-I-Verification complete
```

8.4.5 Wait Option (/W)

Use /W to pause before formatting begins in order to substitute a second volume for the disk you specify in the command line. This is useful for single-disk systems.

After the FORMAT program accepts your command line, it pauses while you exchange volumes. Type Y or any string beginning with Y followed by a carriage return in response to the Continue? prompt when you are ready for formatting to begin. If you type N or any string beginning with N, or CTRL/C, the operation is not performed and the monitor prompt (.) appears. Any other response causes the message to repeat.

When formatting completes, the program pauses again while you replace the original volume. Respond to the Continue? prompt by typing Y or any string beginning with Y followed by a carriage return.

You can combine /W with any other option. The following example formats the diskette in DY: device unit 1 as a single-density diskette.

```
* DY1:/W/S
DY1:/FORMAT-Are you sure?  Y
Mount input volume in _<dev:>;  Continue?  Y
?FORMAT-I-Formatting complete
Mount system volume in _<dev:>;  Continue?  Y
*
```

When you use the /W option, make sure that FORMAT is on the system volume.

8.4.6 No Query Option (/Y)

Use /Y to suppress the Are you sure? confirmation message FORMAT prints before each operation begins. When you use /Y, formatting begins as soon as FORMAT accepts and interprets your command line.

The following example formats the diskette in DY: device unit 1 as a double-density diskette.

```
* DY1:/Y
?FORMAT-I-Formatting complete
*
```

Chapter 9
Logical Disk Subsetting Program (LD)

The logical disk subsetting utility (LD) allows you to define and access logical disks, which is a way of subsetting physical disks. You define a logical disk by associating a logical disk unit number with a file. Once defined, you can use keyboard commands and utility programs to initialize, copy, and utilize these logical disks as if they were physical disks. For example, the COPY/DEVICE command can be used to copy logical disks as well as physical disks.

Disk subsetting is particularly useful when you work with large disks such as RC25, RL01/02, and RK06/07. Large disks such as these often run out of directory entry space before the volume is full. Since each logical disk has its own directory, dividing a physical disk into several logical disks creates more directory entry space. Logical disk subsetting provides a convenient way to group files into logical collections. Logical disk subsetting also allows you to perform some device and file operations more quickly.

9.1 Calling and Terminating LD

To call LD from the system device, first be sure that LD is installed (see the INSTALL command in Chapter 4 of the *RT–11 System User's Guide*). Then respond to the keyboard monitor prompt (.) by typing:

```
. R LD.SYS (RET)
```

When running under the XM monitor, type:

```
. R LDX.SYS (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return, LD prints its current version number and prompts you again for a command string. You can type CTRL/C to terminate LD and return control to the monitor when LD is waiting for input from the console terminal. You must type two CTRL/Cs to terminate LD at any other time.

9.2 LD Command String Syntax

Specify the LD command string in the following general format:

input-specs/options

where:

input-specs	represents the files to be assigned as logical disk units. You can specify up to six input file specifications in a command line. The default file type is .DSK.
options	represents an option from Table 9–1. You must specify at least one option in a command line, and you can specify more than one as long as the operations you specify do not conflict.

9.3 Options

The logical disk subsetting options allow you to mount and dismount logical disk unit numbers and associate them with files; write-lock or write-enable logical disks; and verify and correct logical disk assignments. Table 9–1 summarizes these options. The sections following Table 9–1 describe the LD options and give examples.

Table 9–1: LD Options

Option	Section	Function
/A:ddd	9.3.1	Assigns a logical device name to a logical disk. Must be used with /L.
/C	9.3.2	Verifies all logical disk assignments against the files on the volumes currently mounted.
/L:n	9.3.3	Mounts a logical disk and associates it with a file on a disk, or dismounts a logical disk and disassociates it from a file on a disk.
/R:n	9.3.4	Write-locks a logical disk. When you use /R:n the logical disk you specify has read-only access.
/W:n	9.3.5	Write-enables a logical disk. When you use /W:n, read/write access is allowed for the logical disk you specify.

9.3.1 Assign Logical Device Name Option (/A:ddd)

Use the /A:ddd option with /L to assign a logical device name to a logical disk. The variable ddd represents the logical device name, from one to three characters long, that you want to assign. The first character must be a letter. You can optionally include a colon after the logical device name. After you have assigned a logical device name to a logical disk, you can refer to the logical disk by using the form LDn: or by using the logical device name.

The following command assigns the logical device name VOL to logical disk unit 2 (LD2:) when it is assigned to the file DK:LOGFIL.DSK.

```
* LOGFIL.DSK/L:2/A:VOL
```

9.3.2 Validate Logical Disk Assignments Option (/C)

The /C option validates all logical disk assignments. When you use /C, LD checks the current logical disk assignments against the files on volumes that are mounted.

The /C option is most useful after you have moved or removed files on a volume, or after you have removed a volume from a device. If a logical disk file has moved, LD takes note of the new location so that you can continue to use that logical disk. If you have deleted a logical disk file or the volume containing a logical disk file is no longer mounted, LD disconnects the logical disk assignment. In the case of a volume that you have removed, the disconnection is temporary. You can reestablish the assignment when you remount the volume by using the /C option.

Note that after a squeeze (DUP /S option) or bootstrap operation, the system automatically performs a /C operation to update logical disk assignments.

The /C option must be used alone on a command line. The following command verifies current logical disk assignments.

```
* /C
```

9.3.3 Define Logical Disk Option (/L:n)

The /L:n option mounts a logical disk by associating it with a file on a device, or frees a logical disk number so it can be associated with another file.

Use the following command syntax to mount a logical disk unit number.

filespec/L:n

where:

filespec — represents the file to be associated with a logical disk unit number. The file can reside on either a physical disk or another logical disk.

n — represents the logical disk unit number to associate with the file. After it is mounted, the logical disk is referenced by using the device name LDn:. The variable n must be an integer in the range 0–7.

NOTE

You must be careful to avoid accidentally destroying files while performing logical disk subsetting. LD allows you to assign logical disk unit numbers to both protected and .SYS files, and to write to those files.

To free a logical disk unit number from a file association, type a command with the following syntax.

/L:n

You can mount and dismount several logical disks on the same command line. For example, the following command associates logical disk unit 0 with file MYFILE.DSK on DL0:, logical disk unit 4 with DATFIL.DSK on DY0:, and dismounts logical disk unit 2.

```
* DL0:MYFILE/L:0,DY0:DATFIL.DSK/L:4,/L:2
```

You can also reassign a logical disk unit number by simply specifying the /L option with the same logical disk unit number and a different file name.

9.3.4 Write-Lock Logical Disk Option (/R:n)

Use the /R:n option to write-lock a logical disk. You then have read-only access to that logical disk. The variable n represents the logical disk unit number; n must be an integer in the range 0–7. The default mode is /W (write-enabled).

The following command mounts logical disk unit 3 to JMS.TXT on DY1: and write-locks it.

```
* JMS.TXT/L:3/R:3
```

The next command write-locks logical disk unit 4.

```
* /R:4
```

9.3.5 Write-Enable Logical Disk Option (/W:n)

Use the /W:n option to write-enable a logical disk. You then have read/write access to that logical disk. The variable n represents the logical disk unit number; n must be an integer in the range 0–7. This is the default mode.

The following command mounts logical disk unit 5 to file JMS.DSK on DL0: and write-enables the new logical disk.

```
* DL0:JMS/L:5/W:5
```

Chapter 10
Librarian (LIBR)

The librarian utility program (LIBR) lets you create, update, modify, list, and maintain object library files. It also lets you create macro library files to use with the V03 and later versions of the MACRO–11 assembler.

A library file is a direct access file (a file that has a directory) that contains one or more modules of the same module type. The librarian organizes the library files so that the linker and MACRO–11 assembler can access them rapidly. Each library contains a library header, library directory (or global symbol table, or macro name table), and one or more object modules, or macro definitions. The object modules in a library file can be routines that are repeatedly used in a program, routines that are used by more than one program, or routines that are related and simply gathered together for convenience. The contents of the library file are determined by your needs. An example of a typical object library file is the default system library that the linker uses, SYSLIB.OBJ. An example of a macro library file is SYSMAC.SML, which MACRO uses to process programmed requests.

You access object modules in a library file from another program by making calls or references to their global symbols; you then link the object modules with the program that uses them, producing a single load module (see Chapter 11).

Consult the *RT–11 Software Support Manual* for more information on the internal data structure of a library file.

10.1 Calling and Terminating LIBR

To call the RT–11 librarian from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
.R LIBR (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the console terminal when it is ready to accept a command line.

Type two CTRL/Cs to halt the librarian at any time (or a single CTRL/C to halt the librarian when it is waiting for console terminal input) and return control to the monitor. To restart the librarian, type R LIBR or REENTER in response to the monitor's dot.

10.2 LIBR Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line LIBR accepts.

Specify the LIBR command string in the following general format:

library-filespec[n],list-filespec[n] = input-filespecs/option

where:

library-filespec[n]	represents the library file to be created or updated. The optional argument n represents the number of blocks to allocate for the output file.
list-filespec[n]	represents a listing file for the library's contents. The optional argument n represents the number of blocks to allocate for the listing file.
input-filespecs	represents the input object modules (you can specify up to six input files); it can also represent a library file to be updated.
option	represents an option from Table 10–1.

You specify devices and file names in the standard RT–11 command string syntax, with default file types for object libraries assigned as follows:

Object File	Default File Type
List file	.LST
Library output file	.OBJ
Input file (library or module)	.OBJ

If you do not specify a device, the default device (DK:) is assumed.

Each input file consists of one or more object modules and is stored on a given device under a specific file name and file type. Once you insert an object module into a library file, you no longer reference the module by the name of the file of which it was a part; instead you reference it by its individual module name. You assign this module name with the assembler with either a .TITLE statement in the assembly source program, or with the default name .MAIN. in the absence of a .TITLE statement or the subprogram name for FORTRAN routines. Thus, for example, the input file FORT.OBJ can exist on DY1: and can contain an object module called ABC. Once you insert the module into a library file, reference only ABC (not FORT.OBJ).

The input files normally do not contain main programs but rather subprograms, functions, and subroutines. The library file must never contain a FORTRAN BLOCK DATA subprogram because there is no undefined global symbol to cause the linker to load it automatically.

This section and Section 10.3 explain how to use the librarian to create and maintain object libraries; Section 10.4 describes how to create macro libraries.

10.2.1 Creating a Library File

To create a library file, specify a file name on the output side of a command line.

The following example creates a new library called NEWLIB.OBJ on the default device (DK:). The modules that make up this library file are in the files FIRST.OBJ and SECOND.OBJ, both on the default device.

```
* NEWLIB=FIRST,SECOND
```

10.2.2 Inserting Modules into a Library

Whenever you specify an input file without specifying an associated option, the librarian inserts the input file's modules into the library file you name on the output side of the command string. You can specify any number of input files.

If you include section names (by using /P) in the global symbol table and if you attempt to insert a file that contains a global symbol or PSECT (or CSECT) having the same name as a global symbol or PSECT already existing in the library file, the librarian prints a warning message (see Section 10.3.11 for multiple definition library creation). The librarian does, however, update the library file, ignore the global symbol or section name in error, and return control to the CSI. You can then enter another command string.

Although you can insert object modules that exist under the same name (as assigned by the .TITLE statement), this practice is not recommended because of possible confusion when you need to update these modules (Sections 10.3.8 and 10.3.9 describe replacing and updating).

NOTE

> The librarian performs module insertion, replacement, deletion, merge, and update when creating the library file. Therefore, you must indicate the library file to which the operation is directed on both the input and output sides of the command line, since effectively the librarian creates a new output library file each time it performs one of these operations. You must specify the library file first in the input field.

The following command line inserts the modules included in the files FA.OBJ, FB.OBJ, and FC.OBJ on DY1: into a library file named DXYNEW.OBJ on the default device. The resulting library also includes the contents of library DXY.OBJ.

```
* DXYNEW=DXY,DY1:FA,FB,FC
```

The next command line inserts the modules contained in files THIRD.OBJ and FOURTH.OBJ into the library NEWLIB.OBJ.

```
* NEWLIB,LIST=NEWLIB,THIRD,FOURTH
```

Note that the resulting library contains the original library plus some new modules, and replaces the original library because the same name was used in this example for the input and output library.

10.2.3 Merging Library Files

You can merge two or more library files under one file name by specifying in a single command line all the library files to be merged. The librarian does not delete the individual library files following the merge unless the output file name is identical to one of the input file names.

The command syntax is as follows:

library-filespec = input-filespecs

where:

library-filespec	represents the library file that will contain all the merged files. (If a library file already exists under this name, you must also indicate it in the input side of the command line so that it is included in the merge.)
input-filespec	represents library files to be merged.

Thus, the following command combines library files MAIN.OBJ, TRIG.OBJ, STP.OBJ, and BAC.OBJ under the existing library file name MAIN.OBJ; all files are on the default device DK:. Note that this replaces the old contents of MAIN.OBJ.

```
* MAIN=MAIN,TRIG,STP,BAC
```

The next command creates a library file named FORT.OBJ and merges existing library files A.OBJ, B.OBJ, and C.OBJ under the file name FORT.OBJ.

```
* FORT=A,B,C
```

NOTE

Library files that you combine using PIP are invalid as input to both the librarian and the linker.

10.3 Option Commands and Functions for Object Libraries

You maintain object library files by using option commands. Functions that you can perform include object module deletion, insertion, replacement, and listing of an object library file's contents.

Table 10–1 summarizes the options available for you to use with RT–11 LIBR for object libraries and tells on which command line you must use each option. The following sections, which are arranged alphabetically by option, describe the options in greater detail.

Table 10–1: LIBR Object Options

Option	Command Line	Section	Function
/A	First	10.3.1	Puts all globals in the directory, including all absolute global symbols.
/C	Any but last	10.3.2	Command continuation; allows you to type the input specification on more than one line.
/D	First	10.3.3	Delete; deletes modules that you specify from a library file.
/E	First	10.3.4	Extract; extracts a module from a library and stores it in an OBJ file.
/G	First	10.3.5	Global deletion; deletes global symbols that you specify from the library directory.
/M[:n]	First	10.4.2	Creates a macro library.
/N	First	10.3.6	Names; includes the module names in the directory.
/P	First	10.3.7	P-sect names; includes the program section names in the directory.
/R	First	10.3.8	Replace; replaces modules in a library file. This option must follow the file specification to which it applies.
/U	First	10.3.9	Update; inserts and replaces modules in a library file. This option must follow the file specification to which it applies.
/W	First	10.3.10	Indicates a wide format for the listing file.
/X	First	10.3.11	Allows multiple definitions of global entry points to appear in the library entry point table.
//	First and last	10.3.2	Command continuation; allows you to type the input specification on more than one line.

There is no option to indicate module insertion. If you do not specify an option, the librarian automatically inserts modules into the library file.

10.3.1 Include All Global and Absolute Global Symbols Option (/A)

Use the /A option when you want all the global symbols to appear in the library file's directory. When you use /A, the librarian includes in the directory all absolute global symbols, including those that have a value of 0.

Normally, the librarian includes in the directory only global entry points (labels), and not absolute global symbols.

The following example places all the global symbols from modules MOD1 and MOD2 in the library directory for ALIB.OBJ.

```
* ALIB=MOD1,MOD2/A
```

10.3.2 Command Continuation Options (/C or //)

You must use a continuation option whenever there is not enough room to enter a command string on one line. The maximum number of input files that you can enter on one line is six; you can use the /C option or the // option to enter more.

Type the /C option at the end of the current line and repeat it at the end of subsequent command lines as often as necessary, so long as memory is available; if you exceed memory, an error message prints. Each continuation line after the first command line can contain only input file specifications (and no other options). Do not specify a /C option on the last line of input. If you use the // option, type it at the end of the first input line and again at the end of the last input line.

The following example creates a library file on the default device (DK:) under the file name ALIB.OBJ; it also creates a listing of the library file's contents as LIBLST.LST (also on the default device). The file names of the input modules are MAIN.OBJ, TEST.OBJ, FXN.OBJ, and TRACK.OBJ, all from DY1:.

```
* ALIB,LIBLST=DY1:MAIN,TEST,FXN/C
* DY1:TRACK
```

The next example creates a library file on the default device (DK:) under the name BLIB.OBJ. It does not produce a listing. Input files are MAIN.OBJ from the default device, TEST.OBJ from DL1:, FXN.OBJ from DL0:, and TRACK.OBJ from DY1:.

```
* BLIB=MAIN//
* DL1:TEST
* DL0:FXN
* DY1:TRACK//
```

Another way of writing this command line is:

```
* BLIB=MAIN,DL1:TEST,DL0:FXN//
* DY1:TRACK
* //
```

10.3.3 Delete Option (/D)

The /D option deletes modules and all their associated global symbols from a library file's directory. Since modules are deleted only from the directory

(and not from the object module itself), all modules that were previously deleted are restored whenever you update that library, unless you use /D again to delete them.

When you use the /D option, the librarian prompts:

```
Module name?
```

Respond with the name of the module to be deleted, followed by a carriage return. Continue until you have entered all modules to be deleted. Type a carriage return immediately after the Module name? message to terminate input and initiate execution of the command line.

The following example deletes the modules SGN and TAN from the library file TRAP.OBJ on DY1:.

```
*[DY1:TRAP=DY1:TRAP/D
Module name?  SGN
Module name?  TAN
Module name?
```

The next example deletes the module FIRST from the library LIBFIL.OBJ; all modules in the file ABC.OBJ replace old modules of the same name in the library. It also inserts the modules in the file DEF.OBJ into the library.

```
*. LIBFIL=LIBFIL/D,ABC/R,DEF
Module name?  FIRST
Module name?
```

In the following example, the librarian deletes two modules of the same name from the library file LIBFIL.OBJ.

```
* LIBFIL=LIBFIL/D
Module name?  X
Module name?  X
Module name?
```

10.3.4 Extract Option (/E)

The /E option allows you to extract an object module from a library file and place it in an .OBJ file.

When you specify the /E option, the librarian prints:

```
Global?
```

Respond with the name of the object module you want to extract. If you specify a global name, the librarian extracts the entire module of which that global is a part. Type a carriage return to terminate the prompting for a global.

You cannot use the /E option on the same command line as another option.

The following example extracts the ATAN routine from the FORTRAN library, SYSLIB.OBJ, and stores it in a file called ATAN.OBJ on DX1:.

```
  DX1:ATAN=SYSLIB/E
Global?   ATAN
Global?
```

The next example extracts the $PRINT routine from SYSLIB.OBJ and stores it on DM1: as PRINT.OBJ.

```
  DM1:PRINT=SYSLIB/E
Global?   $PRINT
Global?
```

10.3.5 Delete Global Option (/G)

The /G option lets you delete specific global symbols from a library file's directory.

When you use the /G option, the librarian prints:

```
Global?
```

Respond with the name of the global symbol you want to delete followed by a carriage return; continue until you have entered all globals to be deleted. Type a carriage return immediately after the Global? message to terminate input and initiate execution of the command line.

The following command instructs LIBR to delete the global symbols NAMEA and NAMEB from the directory found in the library file ROLL.OBJ on DK:.

```
  ROLL=ROLL/G
Global?   NAMEA
Global?   NAMEB
Global?
```

The librarian deletes globals only from the directory (and not from the library itself). Whenever you update a library file, all globals that you previously deleted are restored unless you use the /G option again to delete them. This feature lets you recover if you delete the wrong global.

10.3.6 Include Module Names Option (/N)

When you use the /N option on the first line of the command, the librarian includes module names in the directory. The linker loads modules from libraries based on undefined globals, not on module names. The linker also provides equivalent functions by using global symbols and not module names. Normally, then, it is a waste of space and a performance compromise to include module names in the directory.

If you do not include module names in the directory, the MODULE column of the directory listing is blank, unless the module requires a continuation line to print all its globals. A plus (+) sign in the MODULE column indicates continued lines. The /N option is useful mainly when you create a temporary library in order to obtain a directory listing.

If the library does not have module names in its directory, you must create a new library to include the module names. The following example illustrates how to do this. It creates a temporary new library from the current library (by specifying the null device for output) and lists its directory on the terminal. The current library OLDLIB remains unchanged.

```
*↑NL:TEMP,TT:=OLDLIB/N
RT-11 LIBRARIAN V05.00   WED 02-MAR-83 20:36:41
NL:TEMP.OBJ              TUE 02-MAR-83 20:36:40

MODULE          GLOBALS          GLOBALS          GLOBALS

IRAD50          IRAD50           RAD50
JMUL            JMUL
LEN             LEN
SUBSTR          SUBSTR
JADD            JADD
JCMP            JCMP
```

10.3.7 Include P-Section Names Option (/P)

The librarian does not include program section names in the directory unless you use the /P option on the first line of the command. The linker does not use section names to load routines from libraries — in fact, including the names can decrease linker performance. Including program section names also causes a conflict in the library directory and subsequent searches, since the librarian treats section names and global symbols identically.

This option is provided for compatibility with RT–11 V2C. DIGITAL recommends that you avoid using it with later versions of RT–11.

10.3.8 Replace Option (/R)

Use the /R option to replace modules in a library file. The /R option replaces existing modules in the library file you specify as output with the modules of the same names contained in the file(s) you specify as input. In the command string, enter the input library file before the files used in the replacement operation.

If an old module does not exist under the same name as an input module, or if you specify the /R option on a library file, the librarian prints an error message followed by the module name and ignores the replace command.

The /R option must follow each input file name containing modules for replacement.

The following command line indicates that the modules in the file INB.OBJ are to replace existing modules of the same names in the library file TFIL.OBJ. The object modules in the files INA.OBJ and INC.OBJ are to be added to TFIL. All files are stored on the default device DK:.

```
* TFIL=TFIL,INA,INB/R,INC
```

The same operation occurs in the next command as in the preceding example, except that this updated library file is assigned the new name XFIL.

```
* XFIL=TFIL,INA,INB/R,INC
```

10.3.9 Update Option (/U)

The /U option lets you update a library file by combining the insert and replace functions. If the object modules that compose an input file in the command line already exist in the library file, the librarian replaces the old modules in the library file with the new modules in the input file. If the object modules do not already exist in the library file, the librarian inserts those modules into the library. (Note that some of the error messages that might occur with separate insert and replace functions do not print when you use the update function.)

/U must follow each input file that contains modules to be updated. Specify the input library file before the input files in the command line.

The following command line instructs the librarian to update the library file BALIB.OBJ on the default device. First the modules in FOLT.OBJ and BART.OBJ replace old modules of the same names in the library file, or if none already exist under the same names, the modules are inserted. The modules from the file TAL.OBJ are then inserted; an error message prints if the name of a module in TAL.OBJ already exists.

```
* BALIB=BALIB,FOLT/U,TAL,BART/U
```

In the next example, there are two object modules of the same name, X, in both Z and XLIB; these are first deleted from XLIB so that both the modules called X in file Z are correctly placed in the library. Globals SEC1 and SEC2 are also deleted from the directory but automatically return the next time the library XLIB.OBJ is updated.

```
*XLIB=XLIB/D,Z/U/G
Module name? X
Module name? X
Module name?
Global? SEC1
Global? SEC2
Global?
```

10.3.10 Wide Option (/W)

The /W option gives you a wider listing if you request a listing file. The wider listing has six global columns instead of three, as in the normal listing. This is useful if you list the directory on a line printer or a terminal that has 132 columns.

10.3.11 Creating Multiple Definition Libraries Option (/X)

The /X option lets you create libraries that can have more than one definition for a global entry point. These libraries are called multiple definition libraries. They are processed differently from libraries that contain only one definition for each global entry point name that appears in the library's directory (for more information on processing multiple definition libraries, see Chapter 11).

In multiple definition libraries, two library modules may use the same global entry point name, and both definitions may appear in the entry point table (EPT). At least one entry point name should be unique in each module so that you can easily identify it.

When you use the /X option, the librarian does not issue the ?LIBR-W-Invalid insert of AAAAAA message when it encounters a duplicate global symbol name, and the global name will appear in the directory for each module that defines it. In addition, the /X option causes the librarian to turn on the /N option (see Section 10.3.6).

The following example creates the multiple definition library MLTLIB from modules MOD1, MOD2, and MOD3, and lists the library on the terminal. Since MOD3 contains only absolute global symbols, this example must also use the /A option.

```
* MLTLIB,TT:=MOD1,MOD2,MOD3/X/A

RT-11 LIBRARIAN V05.00   THU 11-NOV-82 09:45:31
DK:MLTLIB.OBJ            THU 11-NOV-82 09:45:31

MODULE          GLOBALS          GLOBALS          GLOBALS

MOD1            OMA$R            SWP$             ATP$
MOD2            ATP$             OMA$R            MER$CR
                LBM
MOD3            ATP$             OMA$R            MER$CR
                ENTZ
```

10.3.12 Listing the Directory of a Library File

You can request a listing of the contents of a library file (the global symbol table) by indicating both the library file and a list file in the command line.

Since a library file is not being created or updated, you do not need to indicate the file name on the output side of the command line; however, you must use a comma to designate a null output library file.

The command syntax is as follows:

,LP: = library-filespec

or

,list-filespec = library-filespec

where:

library-filespec	represents the existing library file
LP:	indicates that the listing is to be sent directly to the line printer (or terminal, if you use TT:)
list-filespec	represents a list file of the library file's contents

The following command outputs to DY1: as LIST.LST a listing of all modules in the library file LIBFIL.OBJ, which is stored on the default device.

```
* ,DY1:LIST=LIBFIL
```

The next command sends to the line printer a listing of all modules in the library file FLIB.OBJ, which is stored on the default device.

```
* ,LP:=FLIB
```

Here is a sample section of a large directory listing:

```
* ,TT:=SYSLIB
RT-11 LIBRARIAN V05.00   TUE 02-NOV-82 21:01:01
DK:SYSLIB.OBJ            TUE 02-NOV-82 20:59:47

MODULE          GLOBALS         GLOBALS         GLOBALS

                DCO$            ECO$            FCO$
+               GCO$            RCI$
                DIC$IS          DIC$MS          DIC$PS
+               DIC$SS          $DIVC           $DVC
                ADD$IS          ADD$MS          ADD$PS
+               ADD$SS          SUD$IS          SUD$MS
+               SUD$PS          SUD$SS          $ADD
```

The first line of the listing file shows the version of the librarian that was used and the current date and time. The second line prints the library file name and the date and time the library was created. Each line in the rest of the listing shows only the globals that appear in a particular module. If a module contains more global symbol names than can print on one line, a new line will be started with a plus (+) sign in column 1 to indicate continuation.

If you request a listing of a library file that was created with the /X or /N option, the listing includes module names under the MODULE heading.

10.3.13 Combining Library Option Functions

You can specify two or more library functions in the same command line, with the exception of the /E and /M options, which cannot be specified on the same command line with any other option. The librarian performs functions (and issues appropriate prompts) in the following order:

1. /C or //
2. /D
3. /G
4. /U
5. /R
6. Insertions
7. Listing

Here is an example that combines options:

```
* FILE,LP:=FILE/D,MODX,MODY/R
Module name?  XYZ
Module name?  A
Module name?
```

The librarian performs the functions in this example in the following order:

1. Deletes modules XYZ and A from the library file FILE.OBJ.
2. Replaces any duplicate of the modules in the file MODY.OBJ.
3. Inserts the modules in the file MODX.OBJ.
4. Lists the directory of FILE.OBJ on the line printer.

10.4 Option Commands and Functions for Macro Libraries

The librarian lets you create macro libraries. A macro library works with the V03 or later MACRO–11 assembler to reduce macro search time.

The .MACRO directive produces the entries in the library directory (macro names). LIBR does not maintain a directory listing file for macro libraries; you can print the ASCII input file to list the macros in the library.

The default input file type for macro files is .MAC. The default output file type for macro library files is .MLB.

If you give the library file the same name as one of the input files, the librarian prints the error message: ?LIBR-F-Output and input filnames the same.

The librarian removes all comments from your source input file except for those within a macro (that is, between a .MACRO and .ENDM pair of directives). Because comments take up space during the assembly and in the library, remove them from the macros wherever possible before creating a macro library, if saving space and shortening assembly time are important to you.

Table 10–2 summarizes the options you can use with macro libraries.

The options are explained in detail in the following two sections.

Table 10–2: LIBR Macro Options

Option	Command Line	Section	Meaning
/C	Any but last	10.4.1	Command continuation; allows you to type the input specification on more than one line.
/M[:n]	First	10.4.2	Macro; creates a macro library from the ASCII input file containing .MACRO directives.
//	First and last	10.4.1	Command continuation; allows you to type the input specification on more than one line.

10.4.1 Command Continuation Options (/C or //)

These options are the same for macro libraries as for object libraries. See Section 10.3.2.

10.4.2 Macro Option (/M[:n])

The /M[:n] option creates a macro library file from an ASCII input file that contains .MACRO directives. The optional argument n determines the amount of space to allocate for the macro name directory by representing the number of macros you want the directory to hold. Remember that n is interpreted as an octal number; you must follow n with a decimal point (n.) to indicate a decimal number. One block of library directory space holds 64 macros. The default value for n is 128, enough space for 128 macros, which will use 2 blocks for the macro name table.

The command syntax is as follows:

library-filespec = input-filespec/M[:n]

where:

library-filespec	represents the macro library to be created
input-filespec	represents the ASCII input file that contains .MACRO definitions

The continuation options (/C or //) are the only options you can use with the macro option.

The following example creates the macro library SYSMAC.SML from the ASCII input file SYSMAC.MAC. Both files are on device DK:.

```
* SYSMAC.SML=SYSMAC/M
```

Chapter 11
Linker (LINK)

The linker (LINK) converts object modules to a format suitable for loading and execution. If you have no previous experience with the linker, see the *Introduction to RT–11* for an introductory-level description of the linking process.

To make this chapter easy to use, the description that follows outlines the organization of this chapter.

Section 11.1, Overview of the Linking Process, explains:

- Some of the terms used exclusively in this chapter
- The functions of the linker
- How the linker structures your program to prepare it for execution
- The communication links between modules within your program

Section 11.2, Calling and Terminating the Linker, describes how to invoke the linker from the system device and how to terminate the linker.

Section 11.3, LINK Command String Syntax, describes how to enter a LINK command string. This section also provides a summary of the options you can use in the command string.

Section 11.4, Input and Output, lists and describes the files valid for input to and output from the linker. This section also explains how to use library files, and how the linker processes library files, which you create with the librarian utility (see Chapter 10).

Section 11.5, Creating an Overlay Structure, describes how to design and implement overlay structures for your programs. This section provides detailed descriptions and illustrations of how overlaid programs work and how they reside in memory. This section also explains how to create an overlay structure in extended memory.

Section 11.6, Options, lists and describes the options you can use with the linker.

Section 11.7, Linker Prompts, lists and explains the prompts the linker prints at the terminal after you enter a command line.

11.1 Overview of the Linking Process

A few of the terms used frequently within this chapter, along with their definitions, are listed below. Although the descriptions are brief, you can find more information on these terms in the *Introduction to RT–11* or the *RT–11 Software Support Manual.*

Program section	A named, contiguous unit of code (instructions or data) that is considered an entity and that can be relocated separately without destroying the logic of the program. Also known as a p-sect.
Object module	The primary output of an assembler or compiler, which can be linked with other modules and loaded into memory as a runnable program. The object module is composed of the relocatable machine language code, relocation information, and the corresponding global symbol table defining the use of the symbols within the program. Also known as a module.
Load module	A program in a format ready for loading and executing.
Library file	A file containing one or more relocatable object modules, which are routines that can be incorporated into other programs.
Library module	A module from a library file.
Root segment	The segment of an overlay structure that, when loaded, remains resident in memory during the execution of a program. Also known as the root.
Overlay segment	A section of code treated as a unit that can overlay code already in memory and be overlaid by other overlay segments when called from the root segment or another overlay segment. Also known as an overlay.
Global symbol	A global value or global label.
Low memory	Physical memory from 0 to 28K words.
Extended memory	Physical memory above the 28K word boundary.

11.1.1 What the Linker Does

When the linker processes the object modules, it performs the functions listed below.

- Relocates your program module and assigns absolute addresses
- Links the modules by correlating global symbols that are defined in one module and referenced in another

- Creates the initial control block for the linked program that the GET, R, RUN, SRUN, and FRUN commands use
- Creates an overlay structure, if specified, and includes the necessary run-time overlay handlers and tables
- Searches the library files you specify to locate unresolved global symbols
- Produces a load map, if specified, that shows the layout of the load module
- Produces a symbol table definition file, if specified

The linker requires two passes over the input modules. During the first pass it constructs the symbol table, which includes all program section names and global symbols in the input modules. Next, the linker scans the library files to resolve undefined global symbols. It links only those modules that are required to resolve undefined global symbols. During the second pass, the linker reads in object modules, performs most of the functions listed above, and produces the load module.

The linker runs in a minimal RT–11 system of 16K words of memory; any additional memory is used to facilitate linking and to extend the size of the symbol table. The linker accepts input from any random-access volume on the system; there must be at least one random-access volume (disk, diskette, or DECtape II) for memory image or relocatable format output.

11.1.2 How the Linker Structures the Load Module

When the linker processes the assembled or compiled object modules, it creates a load module in which it has assigned all absolute addresses, has created an absolute section, and has allocated memory for the program sections.

11.1.2.1 Absolute Section – The absolute section is often called the ASECT because the assembler directive .ASECT allows information to be stored there. The absolute section appears in the load map with the name . ABS., and is always the first section in the listing. The absolute section typically ends at address 1000 (octal) and contains the following:

- A system communication area
- Hardware vectors
- A user stack

The system communication area resides in locations 0–377, and contains data the linker uses to pass program control parameters and a memory usage bitmap. Section 11.4.3 provides a detailed description of each location in the system communication area.

The stack is an area that a program can use for temporary storage and subroutine linkage. General register 6, the stack pointer (SP), references the stack.

11.1.2.2 Program Sections — The program sections (p-sects) follow the absolute section. The set of attributes associated with each p-sect controls the allocation and placement of the section within the load module. The p-sect, as the basic unit of memory for a program, has:

- A name by which it can be referenced
- A set of attributes that define its contents, mode of access, allocation, and placement in memory
- A length that determines how much storage is reserved for the p-sect

You create p-sects by using a COMMON statement in FORTRAN, or the .PSECT (or .CSECT) directive in MACRO. You can use the .PSECT (or .CSECT) directive to attach attributes to the section. Note that the attributes that follow the p-sect name in the load map are not part of the name; only the name itself distinguishes one p-sect from another. You should make sure, then, that p-sects of the same name that you want to link together also have the same attribute list. If the linker encounters p-sects with the same name that have different attributes, it prints a warning message and uses the attributes from the first time it encountered the p-sect.

Program Section Attributes

The linker collects from the input modules scattered references to a p-sect and combines them in a single area of the load module. The attributes, which are listed in Table 11–1, control the way the linker collects and places this unit of storage.

The scope-code is meaningful only when you define an overlay structure for the program. In an overlaid program, a global section is known throughout the entire program. Object modules contribute to only one global section of the same name. If two or more segments contribute to a global section, then the linker allocates that global section to the root segment of the program. In contrast to global sections, local sections are only known within a particular program segment. Because of this, several local sections of the same name can appear in different segments. Thus, several object modules contributing to a local section do so only within each segment. An example of a global section is named COMMON in FORTRAN. An example of a local section is the default blank section for each macro routine.

The alloc-code determines the starting address and length of memory allocated by modules that reference a common p-sect. If the alloc-code indicates that such a p-sect is to be overlaid, the linker stores the allocations from each module starting at the same location in memory. It determines the total size from the length of the longest reference to the p-sect. Each module's allocation of memory to a location overwrites that of a previous module. If the alloc-code indicates that a p-sect is to be concatenated, the linker places the allocations from the modules one after the other in the load module; it determines the total allocation from the sum of the lengths of the contributions.

Table 11–1: P-Sect Attributes

Attribute	Value	Explanation
Access-code*	RW	Read/Write – data can be read from, and written into, the p-sect.
	RO	Read Only – data can be read from, but cannot be written into, the p-sect.
Type-code	D	Data – the p-sect contains data, concatenated by byte.
	I	Instruction – the p-sect contains either instructions, or data and instructions, concatenated by word.
Scope-code	GBL	Global – the p-sect name is recognized across segment boundaries. The linker allocates storage in the root for the p-sect from references outside the defining overlay segment. If the p-sect is referenced only in one segment, that p-sect has space allocated in that segment only.
	LCL	Local – the p-sect name is recognized only within each individual segment. The linker allocates storage for the p-sect from references within the segment only.
	SAV	Save – the p-sect name is recognized across segment boundaries. The linker always allocates storage in the root for the p-sect.
Reloc-code	REL	Relocatable – the base address of the p-sect is relocated relative to the virtual base address of the program.
	ABS	Absolute – the base address of the p-sect is not relocated. It is always 0.
Alloc-code	CON	Concatenate – all allocations to a given p-sect name are concatenated. The total allocation is the sum of the individual allocations.
	OVR	Overlay – all allocations to a given p-sect name overlay each other. The total allocation is the length of the longest individual allocation.

* Not supported

Any data (D) p-sect that contains references to word labels must start on a word boundary. You can do this by using the .EVEN assembler directive at the end of each module's concatenated p-sect. (If you do not do this, the program may fail to link, printing the message ?LINK-F-Word relocation error in FILNAM.)

The allocation of memory for a p-sect always begins on a word boundary. If the p-sect has the D (data) and CON (concatenate) attributes, all storage that subsequent modules contribute is appended to the last byte of the previous allocation. This occurs whether or not that byte is on a word boundary. For a p-sect with the I (instruction) and CON attributes, however, all storage that subsequent modules contribute begins at the nearest following word boundary.

The .CSECT directive of MACRO is converted internally by both MACRO and the linker to an equivalent .PSECT with fixed attributes. An unnamed CSECT (blank section) is the same as a blank PSECT with the attributes RW, I, LCL, REL, and CON.

A named CSECT is equivalent to a named PSECT with the attributes RW, I, GBL, REL, and OVR. Table 11–2 shows these sections and their attributes.

Table 11–2: Section Attributes

Section	Access-Code	Type-Code	Scope-Code	Reloc-Code	Alloc-Code
CSECT	RW	I	LCL	REL	CON
CSECT name	RW	I	GBL	REL	OVR
ASECT (. ABS.)	RW	I	GBL	ABS	OVR
COMMON/name/	RW	D	GBL	REL	OVR
VSECT (. VIR.)	RW	D	GBL	REL	CON

The names assigned to p-sects are not considered to be global symbols; you cannot reference them as such. For example:

```
MOV     #PNAME,R0
```

This statement, where PNAME is the name of a section, is invalid and generates the undefined global error message if no global symbol of PNAME exists. A name can be the same for both a p-sect name and a global symbol. The linker treats them separately.

Program Section Order

The linker determines the memory allocation of p-sects by the order of occurrence of the p-sects in the input modules. Table 11–3 shows the order in which p-sects appear for both overlaid and nonoverlaid files.

Table 11–3: P-Sect Order

Nonoverlaid File	Overlaid File
Absolute (. ABS)	Absolute (. ABS)
Blank	Overlay handler ($OHAND)
Named (NAME)	Overlay table ($OTABL)
	Blank
	Named (NAME)

If there is more than one named section, the named sections appear in the order in which they occur in the input files. For example, the FORTRAN compiler arranges the p-sects in the main program module so that the USR can swap over pure code in low memory rather than over data required by the function making the USR call.

If the size of the blank p-sect is 0, it does not appear in the load map.

11.1.3 Global Symbols: Communication Links Between Modules

Global symbols provide the link, or communication, between object modules. You create global symbols with the .GLOBL or .ENABL GBL assembler directive (or with double colon, ::, double equal sign, = =, or = =:).

If the global symbol is defined in an object module (as a label using :: or by direct assignment using = =), other object modules can reference it. If the global symbol is not defined in the object module, it is an external symbol and is assumed to be defined in some other object module. If a global symbol is used as a label in a routine, it is often called an entry point – that is, it is an entry point to that subroutine.

As the linker reads the object modules it keeps track of all global symbol definitions and references. It then modifies the instructions and data that reference the global symbols. The linker always prints undefined globals on the console terminal after pass 1. A list of undefined globals is also included in any load maps you generate.

Table 11–4 shows how the linker resolves global references when it creates the load module.

Table 11–4: Global Reference Resolution

Module Name	Global Definition	Global Reference
IN1	B1 B2	A L1 C1 XXX
IN2	A B1	B2
IN3		B1

In processing the first module, IN1, the linker finds definitions for B1 and B2, and references to A, L1, C1, and XXX. Because no definition currently exists for these references, the linker defers the resolution of these global symbols. In processing the next module, IN2, the linker finds a definition for A that resolves the previous reference, and a reference to B2 that can be immediately resolved.

When all the object modules have been processed, the linker has three unresolved global references remaining: L1, C1, and XXX. A search of the default system library resolves XXX. The global symbols L1 and C1 remain unresolved and are, therefore, listed as undefined global symbols.

The relocatable global symbol, B1, is defined twice and is listed on the terminal as a global symbol with multiple definitions. The linker uses the first definition of such a symbol. An absolute global symbol can be defined more than once without being listed as having multiple definitions, as long as each occurrence of the symbol has the same value.

11.2 Calling and Terminating the Linker

To call the linker from the system device, respond to the dot printed by the keyboard monitor by typing:

```
. R LINK (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. If you enter only a carriage return at this point, the linker prints its current version number.

Type two CTRL/Cs to halt the linker at any time (or a single CTRL/C to halt the linker when it is waiting for console terminal input) and return control to the monitor. To restart the linker, type R LINK or REENTER in response to the monitor's dot.

11.3 Link Command String Syntax

The first command string you enter in response to the linker's prompt has this syntax:

[bin-filespec],[map-filespec],[stb-filespec] = obj-filespec[/option...][,...obj-filespec[/option...]]

where:

bin-filespec	represents the device, file name, and file type to be assigned to the linker's output load module file
map-filespec	represents the device, file name, and file type of the load map output file
stb-filespec	represents the device, file name, and file type of the symbol definition file
obj-filespec	represents an object module, a library file, or a symbol table file, created in a previous link
/option	is one of the options listed in Table 11–6

In each file specification above, the device should be a random-access device, with these exceptions: the output device for the load map file can be any RT–11 device, as can the output device for an .LDA file if you use the /L option. If you do not specify a device, the linker uses default device DK:. Note that the linker load map contains lowercase characters. Use the SET LP LC command to enable lowercase printing if your printer has lowercase characters.

If you do not specify an output file, the linker assumes that you do not want the associated output. For example, if you do not specify the load module and load map (by using a comma in place of each file specification) the linker prints only error messages, if any occur.

Table 11–5 shows the default values for each specification.

Table 11–5: Linker Defaults

	Device	File Name	File Type
Load Module	DK:	None	SAV, REL(/R), LDA(/L)
Load Map	DK: or same as load module	None	MAP
Symbol Definition Output	DK: or same as previous output device	None	STB
Object Module	DK: or same as previous object module	None	OBJ

If you make a syntax error in a command string, the system prints an error message. You can then retype the new command string following the asterisk. Similarly, if you specify a nonexistent file, a warning message occurs; control returns to the CSI, an asterisk prints, and you can reenter the command string.

Table 11–6 lists the options associated with the linker. You must precede the letter representing each option with the slash character. Options must appear on the line indicated if you continue the input on more than one line, but you can position them anywhere on the line. The column titled Command Line lists on which line in the command string the option can appear. (Section 11.6 provides a more detailed explanation of each option.)

Table 11–6: Linker Options

Option Name	Command Line	Section	Explanation
/A	First	11.6.1	Lists global symbols in program sections in alphabetical order.
/B:n	First	11.6.2	Changes the bottom address of a program to n (invalid with /H and /R).
/C	Any but last	11.6.3	Continues input specification on another command line. (You can also use /C with /V and with /O; do not use /C with the // option.)
/D	First	11.6.4	Allows the global symbol you specify to be defined once in each segment that references that symbol. These symbols must be defined in library modules.
/E:n	First	11.6.5	Extends a particular program section in the root to a specific value.
/F	First	11.6.6	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ; this option is provided only for compatibility with previous versions of RT–11.
/G	First	11.6.7	Adjusts the size of the linker's library directory buffer to accommodate the largest multiple definition library directory.
/H:n	First	11.6.8	Specifies the top (highest) address to be used by the relocatable code in the load module. Invalid with /B, /R, /Y and /Q.
/I	First	11.6.9	Extracts the global symbols you specify (and their associated object modules) from the library and links them into the load module.
/K:n	First	11.6.10	Inserts the value you specify (the valid range for n is from 2 to 28.) into word 56 of block 0 of the image file. This option allows you to limit the amount of memory allocated by a .SETTOP request to n K words (decimal). Invalid with /R.
/L	First	11.6.11	Produces a formatted binary output file (invalid for overlaid programs and for foreground links).
/M[:n]	First	11.6.12	Causes the linker to prompt you for a global symbol that represents the stack address or that sets the stack address to the value n. Do not use with /R.
/N	First	11.6.13	Produces a cross-reference in the load map of all global symbols defined during the linking process.

(Continued on next page)

Table 11–6: Linker Options (Cont.)

Option Name	Command Line	Section	Explanation
/O:n	Any but first	11.6.14	Indicates that the program is an overlay structure; n specifies the overlay region to which the module is assigned. Invalid with /L.
/P:n	First	11.6.15	Changes the default amount of space the linker uses for a library routines list.
/Q	First	11.6.16	Lets you specify the base addresses of up to eight root program sections. Invalid with /H or /R.
/R[:n]	First	11.6.17	Produces output in relocatable format and optionally indicates stack size for a foreground job. Invalid with /B, /H, /K, and /L.
/S	First	11.6.18	Makes the maximum amount of space in memory available for the linker's symbol table. (Use this option only when a particular link stream causes a symbol table overflow.)
/T[:n]	First	11.6.19	Cause the linker to prompt you for a global symbol that represents the transfer address or that sets the transfer address to the value n.
/U:n	First	11.6.20	Rounds up the root program section you specify so that the size of the root segment is a whole number multiple of the value you supply (n must be a power of 2).
/V	First	11.6.21	Enables special .SETTOP and .LIMIT features provided by the XM monitor. Invalid with /L.
/V:n[:m]	Any but first	11.6.21	Indicates that an extended memory overlay segment is to be mapped in virtual region n, and optionally in partition m.
/W	First	11.6.22	Directs the linker to produce a wide load map listing.
/X	First	11.6.23	Does not output the bitmap if the code is placed over the bitmap (location 360–377). This option is provided only for compatibility with the RSTS operating system.
/Y:n	First	11.6.24	Starts a specific program section in the root on a particular address boundary. Invalid with /H.
/Z:n	First	11.6.25	Sets unused locations in the load module to the value n.
//	First and last	11.6.3	Allows you to specify command string input on additional lines. Do not use this option with /C.

11.4 Input and Output

Linker input and output is in the form of modules; the linker uses one or more input modules to produce a single output (load) module. The linker also accepts library modules and symbol table definition files as input, and can produce a load map and/or symbol table definition file. The sections that follow describe all valid forms of input to and output from the linker.

11.4.1 Input Object Modules

Object files, consisting of one or more object modules, are the input to the linker. (Entering files that are not object modules may result in a fatal error.) Object modules are created by language translators such as the FORTRAN compiler and the MACRO–11 assembler. The module name item declares the name of the object module (see Section 11.4.4).

The first six Radix—50 characters of the .TITLE assembler directive are used as the name of the object module. These six characters must be Radix—50 characters (the linker ignores any characters beyond the sixth character). The linker prints the first module name it encounters in the input file stream (normally the main routine of the program) on the second line of the map following TITLE:. The linker also uses the first identity label (issued by the .IDENT directive) for the load map. It ignores additional module names.

The linker reads each object module twice. During the first pass it reads each object module to construct a global symbol table and to assign absolute values to the program section names and global symbols. The linker uses the library files to resolve undefined globals. It places their associated object modules in the root if the global symbols in the module are referenced from more than one overlay segment or from the root. If you use the /D option and the global symbols are not referenced from the root, the linker places a copy of the global symbols you specify in each segment that references them. (See Section 11.6.4 for more information on the /D option.) On the second of its two passes, the linker reads the object modules, links and relocates the modules, and outputs the load module.

Symbol table definition files are special object files that can serve as input to LINK anywhere other object files are allowed.

11.4.2 Input Library Modules

The RT–11 linker can automatically search libraries. Libraries consist of library files, which are specially formatted files produced by the librarian program (described in Chapter 10) that contain one or more object modules. The object modules provide routines and functions to aid you in meeting specific programming needs. (For example, FORTRAN has a set of modules containing all necessary computational functions – SQRT, SIN, COS, and so on.) You can use the librarian to create and update libraries. Then you can

easily access routines that you use repeatedly or routines that different programs use. Selected modules from the appropriate library file are linked as needed with your program to produce one load module. Libraries are further described in Chapter 10.

NOTE

> Library files that you combine with the monitor COPY command or with the PIP /U or /B option (described in Chapter 13) are invalid as input to both the linker and the librarian.

You specify libraries in a command string in the same way you specify normal modules; you can include them anywhere in the command string. If you are creating an overlay structure, specify libraries before you specify the overlay structure. Do not specify libraries on the same line as overlay segments. If a global symbol is undefined at the time the linker encounters the library in the input stream, and if a module is included in the library that contains that global definition, then the linker pulls that module from the library and links it into the load image. Only the modules needed to resolve references are pulled from the library; unreferenced modules are not linked.

Modules in one library can call modules from another library; however, the libraries must appear in the command string in the order in which they are called. For example, assume module X in library ALIB calls Y from the BLIB library. To correctly resolve all globals, the order of ALIB and BLIB should appear in the command line as:

```
* Z=B,ALIB,BLIB
```

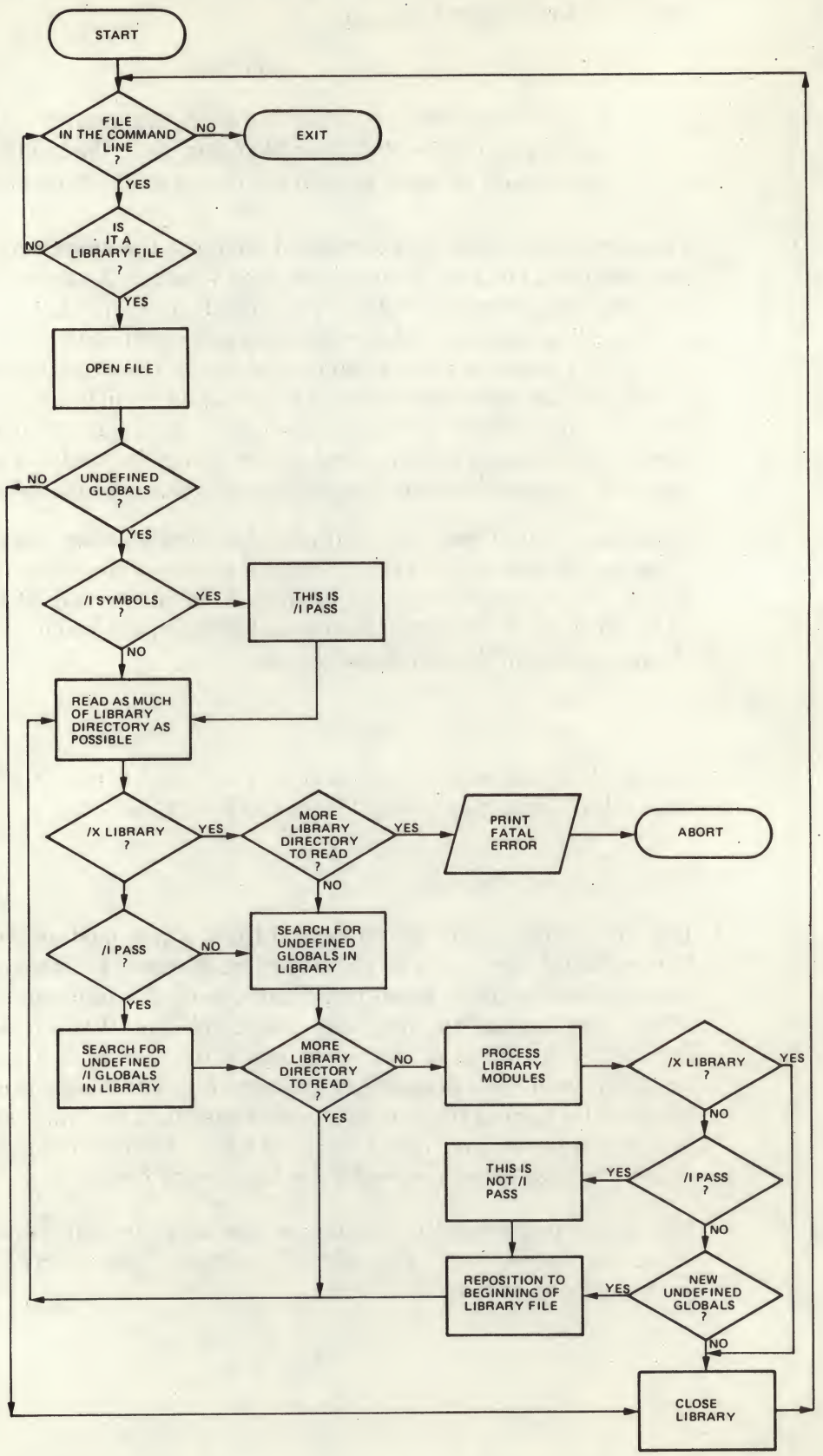
Module B is the root. It calls X from ALIB and brings X into the root. X in turn calls Y, which is brought from BLIB into the root.

Library Module Processing

The linker selectively relocates and links object modules from specific user libraries that were built by the librarian. Figure 11–1 diagrams this general process. During pass 1 the linker processes the input files in the order in which they appear in the input command line. If the linker encounters a library file during pass 1, it takes note of the library in an internal save status block, and then proceeds to the next file. The linker processes only nonlibrary files during the initial phase of pass 1. In the final phase of pass 1 the linker processes only library files. This is when it resolves the undefined globals that were referenced by the nonlibrary files.

The linker processes library files in the order in which they appear in the input command line. The default system library (SY:SYSLIB.OBJ) is always processed last.

Figure 11–1: Library Searches



The search method the linker uses allows modules to appear in any order in the library. You can specify any number of libraries in a link and they can be positioned anywhere, with the exception of forward references between libraries, and they must come before the overlay structure. The default system library, SY:SYSLIB.OBJ, is the last library file the linker searches to resolve any remaining undefined globals.

Some languages, such as FORTRAN, have an Object Time System (OTS) that the linker takes from a library and includes in the final module. The most efficient way to accomplish this is to include these OTS routines (such as NHD, OTSCOM, and V2NS for FORTRAN) in SY:SYSLIB.OBJ. See the *RT–11 Installation Guide* for details on how to do this.

Libraries are input to the linker the same way as other input files. Here is a sample LINK command string:

```
* TASK01,LP:=MAIN,MEASUR
```

This causes program MAIN.OBJ to be read from DK: as the first input file. Any undefined symbols generated by program MAIN.OBJ should be satisfied by the library file MEASUR.OBJ specified in the second input file. The linker tries to satisfy any remaining undefined globals from the default library, SY:SYSLIB.OBJ. The load module, TASK01.SAV, is stored on DK: and a load map prints on the line printer.

Multiple Definition Libraries

In addition to the libraries explained so far, LINK processes multiple definition libraries. DIGITAL does not recommend that you use this type of library in normal situations; its primary purpose is to provide special functions for RSTS. These libraries differ from other libraries in that they can contain more than one definition for a given global. You specify multiple definition libraries in the command line the same way you specify normal libraries. Modules that LINK obtains from multiple definition libraries always appear in the root.

It is useful to be aware of the differences between processing normal and multiple definition libraries. When you include modules from a multiple definition library, LINK has to store that library's directory in an internal buffer. A library's directory is often called an entry point table (EPT). If a library EPT is too large to fit into the internal buffer, LINK prints a message instructing you to use the /G option. The /G option changes the buffer's size to accommodate the largest EPT of all the multiple definition libraries you are using. Use the /G option only when LINK indicates it is required.

When a global symbol from a multiple definition library matches an undefined global, LINK removes from the undefined global list all other globals defined in the same library. LINK does this before it processes the library module. Thus, two modules with identical globals will not appear in the linked module.

NOTE

The order of modules in multiple definition libraries is very important and will affect which modules LINK uses. The increased EPT size (due to duplicate entries, in addition to module name entries) will also slow LINK down.

11.4.3 Output Load Module

The primary output of the linker is a load module that you can run under RT–11. The linker creates as a load module a memory image file (file type of .SAV) for use under a single-job system or as the background job under the FB monitor; save images can also be run as virtual foreground jobs under the XM monitor. If you need to execute a program in the foreground, use the /R option to produce a relocatable format (file type of .REL) foreground load module. The linker can produce an absolute load module (file type of .LDA) if you need to load the module with the Absolute Loader. See the *RT–11 Software Support Manual* for more details on these formats.

The load module for a memory image file is arranged as follows:

Root Segment	Overlay Segments (optional)

For a relocatable image file the load modules are arranged as follows:

Root Segment	Overlay Segments (optional)	Relocation information for root and overlay segments

The first 256-word block of the root segment (main program) contains the memory usage bitmap and the locations the linker uses to pass program control parameters. The memory usage bitmap outlines the blocks of memory that the load module uses; it is located in locations 360 through 377.

Table 11–7 lists the parameters that appear in the absolute block, the addresses the parameters occupy, and the conditions under which they are set.

The linker stores default values in locations 40, 42, and 50, unless you use options to specify otherwise. The /T option affects location 40, for example, and /M affects location 42. You can also use the .ASECT directive to change the defaults. The overlay bit is located in the job status word. LINK automatically sets this bit if the program is overlaid. Otherwise, the linker initially sets location 44 to 0. Location 46 also contains zero unless you specify another value by using the .ASECT directive.

You can assign initial values to memory locations 0–476 (which include the interrupt vectors and system communication area) by using an .ASECT assembler directive. The values appear in block 0 of the load module, but

Table 11–7: Absolute Block Parameters

Address	Parameter	When Set
0	Identification of a program that was created with /V option	Only with /V
2	Highest virtual memory address used by the program	Only with /V
14,16	(XM only) BPT trap	Only with /R
20,22	(XM only) IOT trap	Only with /R
34,36	TRAP vector	Only with /R
40	Start address of program	Always
42	Initial setting of SP (stack pointer)	Always
44	Job Status Word (overlay bit set by LINK)	Always
46	USR swap (set by user) address; (0 implies normal location)	Always
50	Highest memory address used by the program (high limit)	Always
52	Size of root segment in bytes	Only with /R
54	Stack size in bytes (value with /R or default 128)	Only with /R
56	Size of overlay region in bytes	Only with /R
60	Identification of a file in relocatable (.REL) format	Only with /R
62	Relative block number for start of relocation information	Only with /R
64	Start address of overlay table	With /O or /V
66	Start of virtual overlay segment information in overlay handler tables	Only with /V
360–377	Memory usage bitmap	Always, except with /X or /L

there are restrictions on the use of .ASECT directives in this region. You should not modify location 54 or locations 360–377 because the memory usage map is passed in those locations. In addition, for foreground links, modifications of words 52–62 are not permitted because additional parameters are passed to the FRUN command in those locations.

You can use an .ASECT directive to set any location that is not restricted, but be careful if you change the system communication area. The program itself must initialize restricted areas, such as locations 360–377. There are no restrictions on .ASECT directives if the output format is LDA.

Locations in addresses 0–476 might not be loaded at execution time, even though your program uses an .ASECT to initialize them. For background programs, this is because the R, RUN, and GET commands do not load addresses that are protected by the monitor's memory protection map. For foreground programs, the FRUN command loads only locations 14–22 and 34–50 and ignores all other low memory locations. To initialize a location at run time, use the .PROTECT programmed request. If it is successful, follow it with a MOV instruction to modify the location.

11.4.4 Output Load Map

The linker can produce a load map following the completion of the initial pass. This map, shown in Figure 11–2, diagrams the layout of memory for the load module.

The load map lists each program section that is included in the linking process. The line for a section includes the name and low address of the section and its size in bytes. The rest of the line lists the program section attributes, as shown in Table 11–2. The remaining columns contain the global symbols found in the section and their values.

Figure 11–2: Sample Load Map

```
1    RT-11  LINK   V06.01        Load Map            Friday 14-Jan-83 11:25        Page 1
2    TEST   .SAV        Title:   TEST      Ident:
3
4    Section.   Addr    Size     Global    Value     Global    Value     Global    Value
5
6    . ABS.     000000  001000 = 256.      words     (RW,I,GBL,ABS,OVR)
7               001000  000200 =  64.      words     (RW,I,LCL,REL,CON)
8    TEST       001200  000174 =  62.      words     (RW,I,LCL,REL,CON)
9                                START     001200    EXIT      001240
10
11   Transfer address = 001200, High limit = 001372 = 381.  words
```

Table 11–8 describes each line in the sample load map above.

The map begins with the linker version number, followed by the date and time the program was linked. The second line lists the file name of the program, its title (which is determined by the first module name record in the input file), and the first identification record found. The absolute section is always shown first, followed by any nonrelocatable symbols. The modules located in the root segment of the load module are listed next, followed by those modules that were assigned to overlays in order by their region number (see Section 11.5). Any undefined global symbols are then listed. The map ends with the transfer address (start address) and high limit of relocatable code in both octal bytes and decimal words. If you use the /N option, a

Table 11–8: Line-by-Line Sample Load Map Description

Line	Contents
1	Load map header.
2	Program name, program title (.MAIN. default) and identity (default is blank).
4	P-sect description header. Section indicates the p-sect name; Addr indicates the p-sect start address; Size indicates p-sect length in octal bytes; Global and Value list the p-sect globals and their associated octal values.
6	Absolute p-sect, . ABS. This line includes the absolute p-sect's start address, length and attributes (for a complete descr iption of these abbreviations, see Table 11–1). The linker always includes a . ABS. p-sect in the link.
7	Unnamed p-sect. This p-sect appears in the load map after the absolute p-sect. For overlaid programs, the unnamed p-sect appears in the load map after the overlay table p-sect (see Figure 11–11).
8–9	TEST p-sect. Line 9 lists TEST's two globals, START and EXIT, with their associated values.
11	Transfer address indicates the address in memory where the program starts. High limit indicates the last address used by the program. The number of words in the program appears last.

cross-reference of all global symbols defined during the linking process follows the transfer address line. See Figure 11–14 for a sample and description of a global cross-references table.

NOTE

The load map does not reflect the absolute addresses for a REL file that you create to run as a foreground job; you must add the base relocation address determined at FRUN time to obtain the absolute addresses. The linker assumes a base address of 1000.

For example, assume the FRUN command is used to run the program TEST:

```
. FRUN TEST/P
Loaded at 127276
```

The /P option causes FRUN to print the load address, which is 127276 in this example. To calculate the actual location in memory of any global in the program, first subtract 1000 from that global's value. (The value 1000 represents the base address assigned by the linker. This offset is not used at load time.) Then add the result to the load address determined with /P. The final result represents the absolute location of the global.

11.5 Creating an Overlay Structure

The ability of RT–11 to handle overlays gives you virtually unlimited memory space for an assembly language or FORTRAN program. A program using overlays can be much larger than would normally fit in the available memory space, since portions of the program reside on a storage device such as disk. To utilize this capability, you must define an overlay structure for your program.

Prior to Version 4, RT–11 permitted overlays to be placed only in low memory. Now you can place them in extended memory, too, if you run your program on a system that has an extended memory configuration and XM monitor. Overlays that reside in low memory are called low memory overlays, and those in extended memory are called extended memory overlays.

Section 11.5.1, Low Memory Overlays, describes low memory overlays in general and shows how to define a low memory overlay structure for your program. Section 11.5.2, Extended Memory Overlays, deals specifically with extended memory overlays, and shows how to define an overlay structure that has either extended memory overlays only or both extended memory and low memory overlays.

Read 11.5.1 before reading 11.5.2, because much of the information contained in the first section applies to the second section.

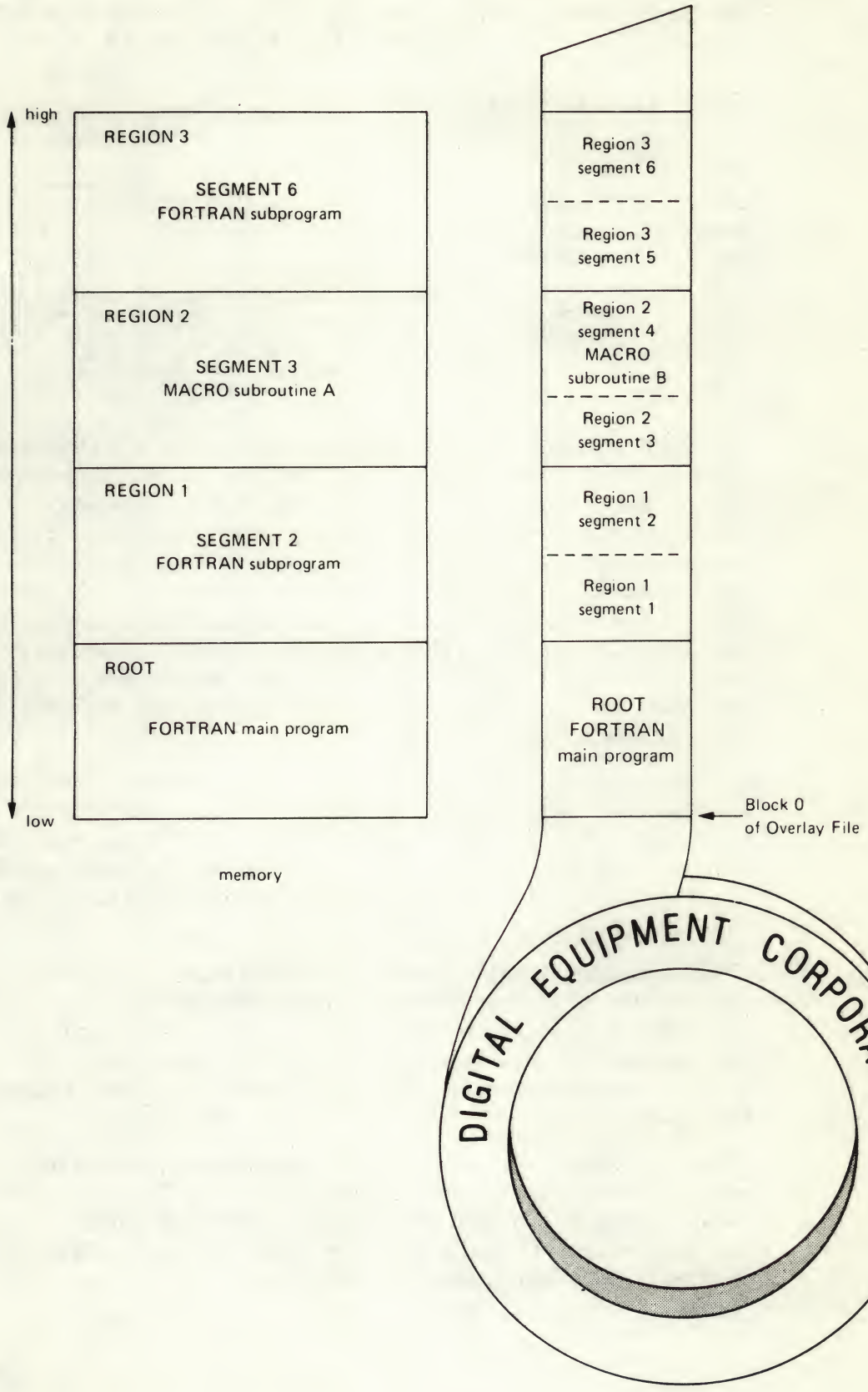
11.5.1 Low Memory Overlays

An overlay structure divides a program into segments. For each overlaid program there is one root segment and a number of overlay segments. Each overlay segment is assigned to a particular area of available memory called an overlay region. More than one overlay segment can be assigned to a given overlay region. However, each region of memory is occupied by one (and only one) of its assigned segments at a time. The other segments assigned to that region are stored on disk, diskette, or DECtape II. They are brought into memory when called, replacing (overlaying) the segment previously stored in that region. The root segment, on the other hand, contains those parts of the program that must always be memory resident. Therefore the root is never overlaid by another segment.

Figure 11–3 diagrams an overlay structure for a FORTRAN program. The main program is placed in the root segment and is never overlaid. The various MACRO subroutines and FORTRAN subprograms are placed in overlay segments. Each overlay segment is assigned to an overlay region and stored on DECtape until called into memory. For example, region 2 is shared by the MACRO subroutine A currently in memory and the MACRO subroutine B in segment 4. When a call is made to subroutine B, segment 4 is brought into region 2 of memory, overlaying or replacing segment 3.

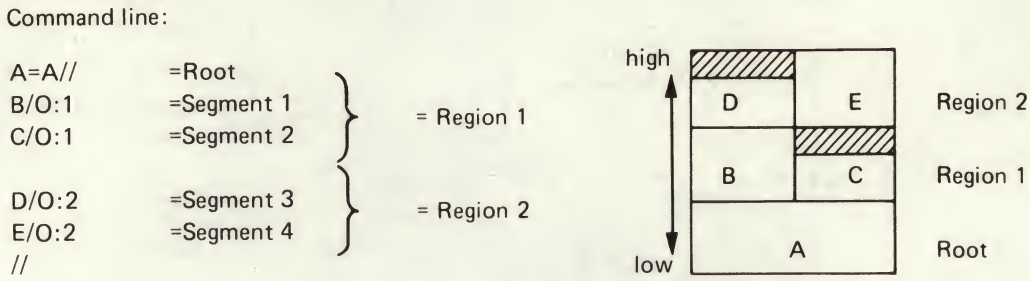
The overlay file, shown on the DECtape in Figure 11–3, is created by the linker when you specify an overlay structure. The overlay file contains at all times a copy of the root segment and each overlay segment, including those overlay segments currently in memory.

Figure 11–3: Sample Overlay Structure for a FORTRAN Program



You specify an overlay structure to the linker by using the /O option (see Figure 11–4). To specify an overlay structure that uses extended memory, use the /V option (see Section 11.5.2 for a discussion of extended memory overlays). This option is described fully in Section 11.5.2.4.

Figure 11–4: Overlay Scheme



The linker calculates the size of any region to be the size of the largest segment assigned to that region. Thus, to reduce the size of a program (that is, the amount of memory it needs), you should first concentrate on reducing the size of the largest segment in each region. The linker delineates the overlay regions you specify, and prefaces your program with the run-time overlay handler code shown in Figure 11–5. The linker also sets up links between the overlay handler and program references to routines that reside in overlays. When, at run time, a reference is made to a section of your program that is not currently in memory, these links cause an overlay to be read into memory. The overlay segment containing the referenced code becomes resident.

There is no special formula for creating an overlay structure. You do not need a special code or function call. However, some general guidelines must be followed. For example, a FORTRAN main program must always be placed in the root segment. This is true also for a global program section (such as a named COMMON block) that is referenced by more than one overlay segment.

The assignment of region numbers to overlay segments is crucial. Segments that overlay each other (have the same region number) must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region. Segments that need to be memory resident simultaneously must be assigned to different regions.

When you make calls to routines or subprograms that are in overlay segments, the entire return path must be in memory. This means that from an overlay segment you cannot call a routine that is in a different segment of the same region. If this is done, the called routine overlays the segment making the call, and destroys the return path.

Figure 11–5: Run-Time Overlay Handler – Low Memory

```
.TITLE  OHANDL LOW MEMORY OVERLAY HANDLER
.SBTTL  THE RUN-TIME OVERLAY HANDLER
.ENABL GBL

;+
; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
; LINKER WHENEVER LOW MEMORY OVERLAYS ARE REQUESTED BY THE USER.
; THE RUN-TIME LOW MEMORY OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:
;
;          JSR      R5,$OVRH            ;CALL TO COMMON CODE FOR LOW MEMORY OVERLAYS
;          .WORD    <OVERLAY #*6>       ;# OF DESIRED SEGMENT
;          .WORD    <ENTRY ADDR>        ;ACTUAL CORE ADDRESS (VIRTUAL ADDRESS)
;
; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
; OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO A LOW MEMORY OVERLAY SEGMENT.
; ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO BE
; REFERENCES TO THE APPROPRIATE DUMMY ROUTINE.  EACH OVERLAY SEGMENT
; IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE.  AN
; OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY.  ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.
;
; THERE IS ONE WORD PREFIXED TO EVERY OVERLAY REGION THAT IDENTIFIES THE
; SEGMENT CURRENTLY RESIDENT IN THAT OVERLAY REGION. THIS WORD IS AN INDEX
; INTO THE OVERLAY TABLE, AND POINTS AT THE OVERLAY SEGMENT INFORMATION.
;
; UNDEFINED GLOBALS IN THE OVERLAY HANDLER MUST BE NAMED "$OVDF1" TO
; "$OVDFn" SUCH THAT A RANGE CHECK MAY BE DONE BY LINK TO DETERMINE IF
; THE UNDEFINED GLOBAL NAME IS FROM THE OVERLAY HANDLER.  A CHECK IS
; DONE ON THE .RAD50 CHARACTERS "$OV", AND THEN A RANGE CHECK IS DONE ON
; THE .RAD50 CHARACTERS "DF1" TO "DFn".  THESE GLOBAL SYMBOLS DO NOT APPEAR
; ON LINK MAPS, SINCE THEIR VALUE IS NOT KNOWN UNTIL AFTER THE MAP HAS BEEN
; PRINTED.  CURRENTLY $OVDF1 TO $OVDF5 ARE IN USE.
;
; GLOBAL SYMBOLS O$READ AND O$DONE ARE USEFUL WHEN DEBUGGING OVERLAID
; PROGRAMS.
;
;       O$READ:: WILL APPEAR IN THE LINK MAP AND LOCATES THE .READW
;       STATEMENT IN THE OVERLAY HANDLER.
;
;       O$DONE:: WILL APPEAR IN THE LINK MAP AND LOCATES THE FIRST
;       INSTRUCTION AFTER THE .READW IN THE OVERLAY HANDLER.
;-

.MCALL   .READW,..V1..
         .V1..              ;V1 FORMAT

.SBTTL   OVERLAY HANDLER CODE

.PSECT   $OHAND,GBL

.ENABL   LSB

; $OVRH IS THE ENTRY POINT TO THE OVERLAY HANDLER

         .RAD50   /OVR/                ;THIS KEEPS HANDLER THE SAME SIZE AS V03
$OVRH::  MOV      R0,-(SP)             ;/O OVERLAY ENTRY POINT
         MOV      R1,-(SP)             ;SAVE REGISTERS
         MOV      R2,-(SP)

1$:
         BR       5$                   ;FIRST CALL ONLY * * *
;        MOV      @R5,R1               ;PICK UP OVERLAY NUMBER
         ADD      #$OVTAB-6,R1         ;CALC TABLE ADDR
         MOV      (R1)+,R2             ;GET FIRST ARG. OF OVERLAY SEG. ENTRY
2$:      CMP      (R5)+,@R2            ;IS OVERLAY ALREADY RESIDENT?
         BEQ      3$                   ;YES, BRANCH TO IT
```

(Continued on next page)

```
;+
; THE .READW ARGUMENTS ARE AS FOLLOWS:
; CHANNEL NUMBER, CORE ADDRESS, LENGTH TO READ, RELATIVE BLOCK ON DISK.
; THESE ARE USED IN REVERSE ORDER FROM THAT SPECIFIED IN THE CALL.
;-

O$READ:: .READW   17,R2,@R1,(R1)+      ;READ FROM OVERLAY FILE
O$DONE:: BCS      4$
3$:      MOV      (SP)+,R2             ;RESTORE USERS REGISTERS
         MOV      (SP)+,R1
         MOV      (SP)+,R0
         MOV      @R5,R5               ;GET ENTRY ADDRESS
         RTS      R5                   ;ENTER OVERLAY ROUTINE AND RESTORE USER'S R5

4$:      EMT      376                  ;SYSTEM ERROR 10 (OVERLAY I/O)
         .BYTE    0,373

5$:      MOV      #11501,1$            ;RESTORE SWITCH INSTR (MOV _@R5,R1)
         MOV      $ODF1,R1             ;START ADDR FOR CLEAR OPERATION
6$:      CLR      (R1)+                ;CLEAR ALL OVERLAY REGIONS
         CMP      R1,$ODF2             ;DONE?
         BLO      6$                   ;LO -> NO, REPEAT
         BR       1$                   ;AND RETURN TO CALL IN PROGRESS

$ODF1::  .WORD    $OVDF1               ;HIGH ADDR OF ROOT SEGMENT + 2 (NXT AVAIL)
$ODF2::  .WORD    $OVDF2               ;HIGH ADDRESS OF /O OVERLAYS + 2 (NXT AVAIL)

.DSABL   LSB

.SBTTL   $OVTAB   OVERLAY TABLE

;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->        $OVTAB:
;                  .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O OVERLAYS
;                  DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
;-

.PSECT   $OTABL,D,GBL,OVR

$OVTAB:

.END
```

Figure 11–6 illustrates a sample set of subroutine calls and return paths. In the example, solid lines represent valid subroutine calls and dotted lines represent invalid calls.

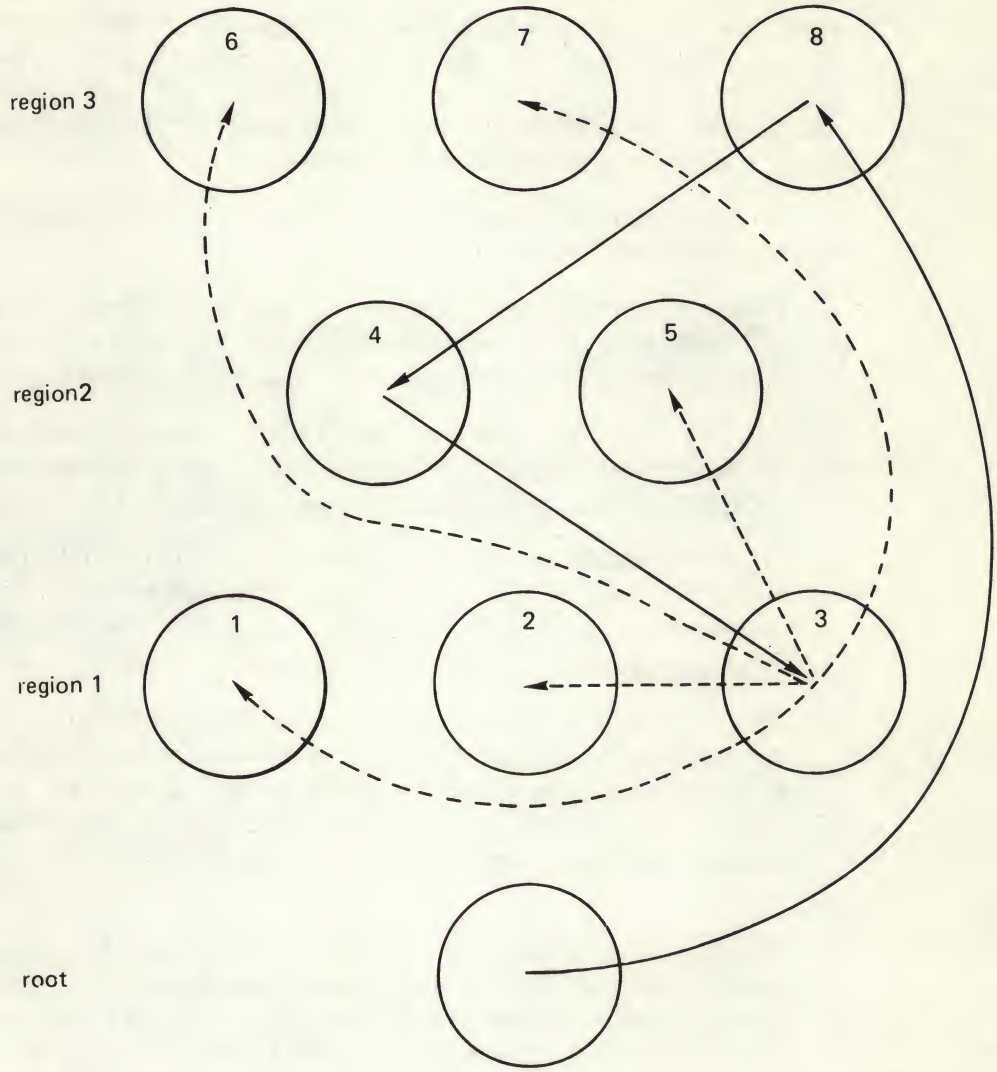
Suppose the following subroutine calls were made:

1. The root calls segment 8
2. Segment 8 calls segment 4
3. Segment 4 calls segment 3

Segment 3 can now call any of the following, in any order:

Itself
Segment 4
Segment 8
The root

Figure 11–6: Sample Subroutine Calls and Return Paths



These segments and the root, of course, are all currently resident in memory.

Segment 3 cannot call any of the following segments because this would destroy its return path:

Segments 2 and 1
Segment 5
Segments 6 and 7

Look at what might happen if one of these invalid calls is made. Assume that segments 3, 4, and 5 all contain MACRO subroutines. Suppose segment 4 calls segment 3 and segment 3 in turn calls segment 5. Segment 5 is not resident in region 2, so an overlay read-in occurs: segment 5 is read into

memory, thus destroying the memory-resident copy of segment 4. The subroutine in segment 5 executes and returns control to segment 3. Segment 3 finishes its task and tries to return control to segment 4. Segment 4, however, has been replaced in memory by segment 5. Segment 4 cannot regain control and the program loops indefinitely, traps, or random results occur.

The guidelines already mentioned and some additional rules for creating overlay structures are summarized below.

1. SYSLIB must be present to create an overlay structure because it contains the overlay handler.

2. Overlay segments assigned to the same region must be logically independent; that is, the components of one segment cannot reference the components of another segment assigned to the same region.

3. The root segment contains the transfer address, stack space, impure variables, data, and variables needed by many different segments. The FORTRAN main program unit must be placed in the root segment.

4. A global program section (such as a named COMMON block or a .PSECT with the GBL attribute) that is referenced in more than one segment is placed in the root segment by the linker. This permits common access across the different segments.

5. Object modules that are automatically acquired from a library file will automatically be placed in an overlay segment, so long as that library module is referenced only by that segment. If a library module is referenced by more than one segment, LINK places that library module in the root unless you use the /D option. See Section 11.6.4 for more details on /D.

   Do not specify a library file on the same command line as an overlay segment. You must specify all library modules before specifying any overlay modules. Link places in the root any modules from a multiple definition library and any modules included with the /I option.

6. All COMMON blocks that are initialized with DATA statements must be similarly initialized in the segment in which they are placed.

7. When you make calls to overlay segments, the entire return path to the calling routine must be in memory. (With extended memory overlays, the entire return path must be mapped. See Section 11.5.2.) This means you should take the following points into account:

   a. You can make calls with expected return (as from a FORTRAN main program to a FORTRAN or MACRO subroutine) from an overlay segment to entries in the same segment, the root segment, or to any other segment, so long as the called segment does not overlay in memory part of your return path to the main program.

   b. You can make jumps with no expected return (as in a MACRO program) from an overlay segment to any entry in the program with one exception: you can not make such a jump to a segment if the called

segment will overlay an active routine (that is, a routine whose execution has begun, but not finished, and that will be returned to) in that region.

c. Calls you make to entries in the same region as the calling routine must be entirely within the same segment, not within another segment in the same region.

8. You must make calls or jumps to overlay segments directly to global symbols defined in an instruction p-sect (entry points). For example, if ENTER is a global tag in an overlay segment, the first of the following two commands is valid, but the second is not:

```
JMP ENTER      ;VALID
JMP ENTER+6    ;INVALID
```

9. You can use globals defined in an instruction p-sect (entry points) of an overlay segment only for transfer of control and not for referencing data within an overlay segment. The assembler and linker cannot detect a violation of this rule so they issue no error. However, such a violation can cause the program to use incorrect data. If you reference these global symbols outside of their defining segment, the linker resolves them by using dummy subroutines of four words each in the overlay handler. Such a reference is indicated on the load map by a @ following the symbol.

10. The linker directly resolves symbols that you define in a data p-sect. It is your responsibility to load the data into memory before referencing a global symbol defined in a data section.

11. You cannot use a section name to pass control to an overlay because it does not load the appropriate segment into memory. For example, JSR PC,OVSEC is invalid if you use OVSEC as a .CSECT name in an overlay. You must use a global symbol to pass control from one segment to the next.

12. In the linker command string, specify overlay regions in ascending order.

13. Overlay regions are read-only. Unlike USR swapping, an overlay handler does not save the segment it is overlaying. Any tables, variables, or instructions that are modified within a given overlay segment are reinitialized to their original values in the SAV or REL file if that segment has been overlaid by another segment. You should place any variables or tables whose values must be maintained across overlays in the root segment.

14. Your program cannot use channel 17 (octal) because overlays are read on that channel.

15. MACRO and FORTRAN directly resolve all global symbols that are defined in a module. If LINK moves the p-sect where they are defined from an overlay segment to the root, LINK will not generate an overlay table entry for those symbols.

Refer to the *RT–11/RSTS/E FORTRAN IV User's Guide* for additional information.

The absolute section (. ABS.) never takes part in overlaying in any way. It is part of the root and is always resident.

This set of rules applies only to communications among the various modules that make up a program. Internally, each module must only observe standard programming rules for the PDP–11 (as described in the *PDP–11 Processor Handbook* and in the FORTRAN and MACRO–11 Language Reference Manuals).

Note that the condition codes set by your program are not preserved across overlay segment boundaries.

The linker provides overlay services by including a small resident overlay handler in the same file with your program to be used at program run time. The linker inserts this overlay handler plus some tables into your program beginning at the bottom address. The linker then moves your program up in memory to make room for the overlay handler and tables, if necessary. The handler is stored in SYSLIB. This scheme is diagrammed in Figure 11–7.

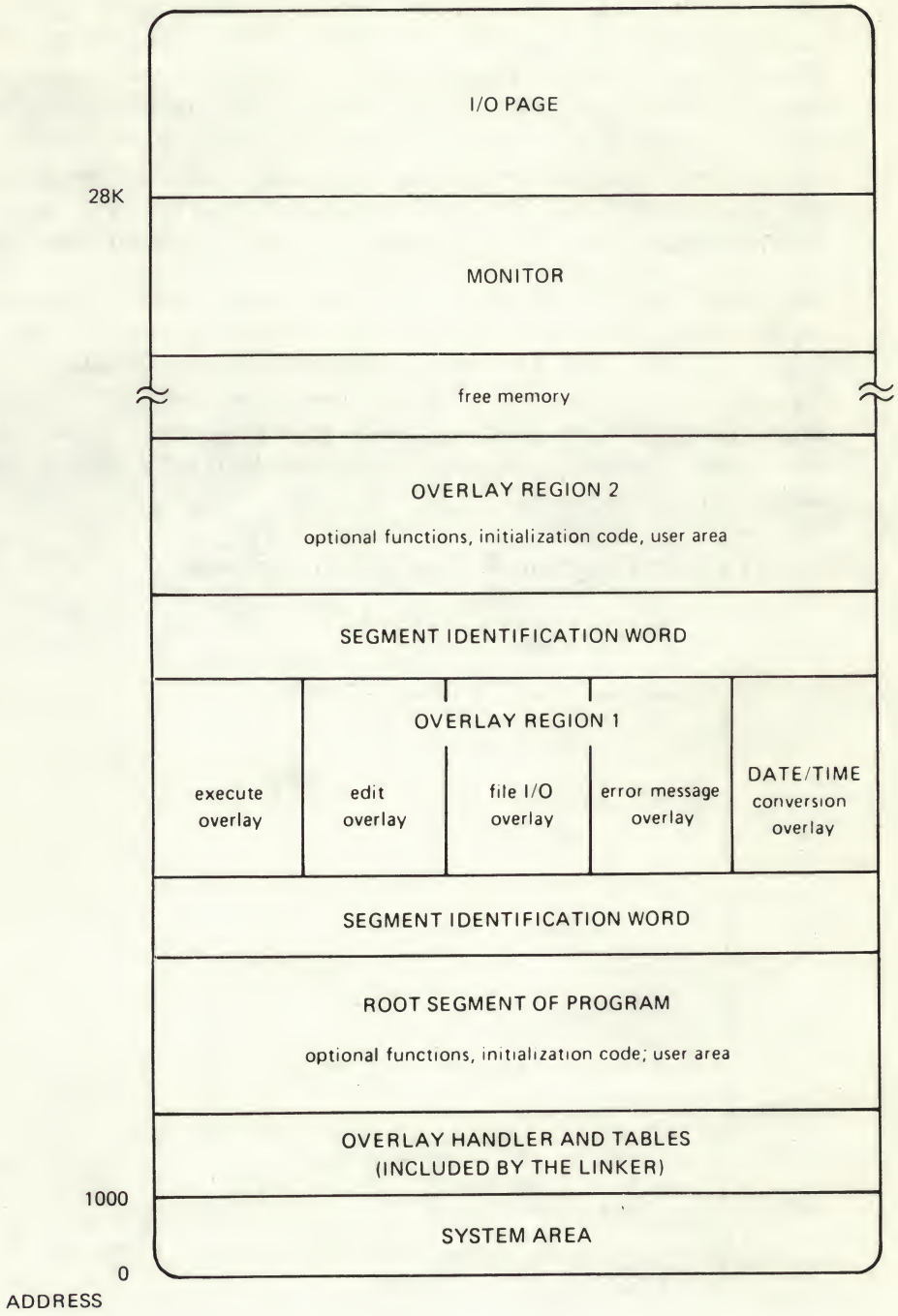
11.5.2 Extended Memory Overlays

You can use LINK to create an overlay structure for your program that uses extended memory. Although you need a hardware configuration that includes a memory management unit to run a program that has overlays in extended memory, you can link it on any RT–11 system. Read Section 11.5.1, Low Memory Overlays, before reading this section — much of the information contained in that section applies to extended memory overlays as well.

Usually, you can convert an overlaid program to use extended memory without modifying the code. The extended memory overlay handler and the keyboard monitor include all the programmed requests necessary to access extended memory (see the *RT–11 Software Support Manual* for details on extended memory restrictions). The overlay tables also include additional data used by these requests, so you can access extended memory automatically without using extended memory programmed requests in your program. Refer to the *RT–11 Software Support Manual* for more information on extended memory.

The extended memory overlay structure is different from the low memory overlay structure in that extended memory overlays can reside concurrently in extended memory. This allows for speedier execution because, once read in, your program requires fewer I/O transfers with the auxiliary mass storage volume. If all program data is resident, and the program is loaded, the program may be able to run without an auxiliary mass storage volume. However, you must observe the same restrictions with extended memory overlays that apply to low memory overlays, especially regarding return paths. This section describes how to create a program with overlays in extended memory and ends with an example of such a program.

Figure 11–7: Memory Diagram Showing BASIC Link with Overlay Regions



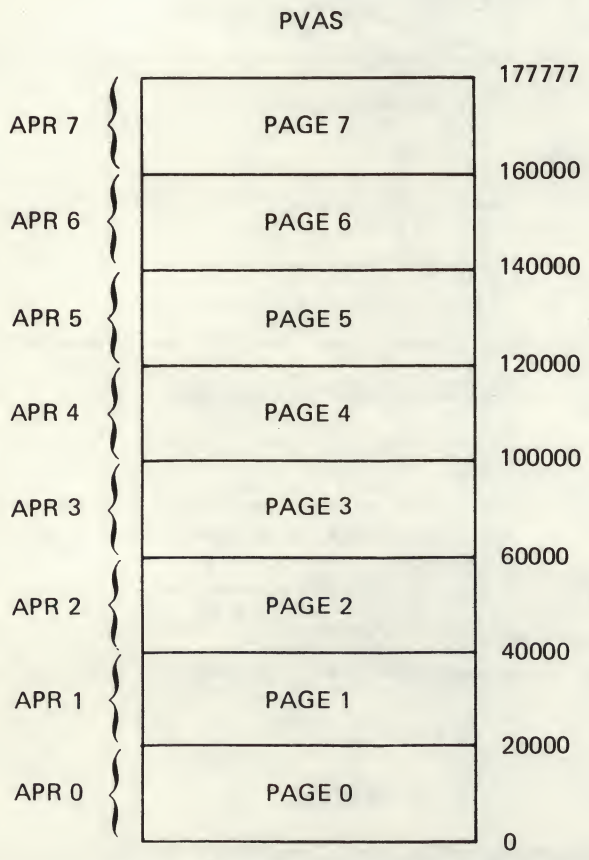
NOTE

Overlays that reside in extended memory can contain impure data, but impure data is not automatically initialized each time a new overlay segment maps over a segment that contains impure data.

11.5.2.1 Virtual Address Space – When you set up an extended memory overlay structure, you set it up as though you had locations 0 to 177777 (that is, 32K words of memory) available for your use. Physically, not all these locations are available to you in low memory; your program's absolute section resides, typically, in locations 0 to 500, and the monitor takes up a good deal of memory starting at location 160000, going downward. Also, the computer sets aside addresses 160000 to 177777 for the I/O page. But, because of memory management, you can structure your program as though you had all 32K words of memory for your use. This space is called the program virtual address space (PVAS). The memory management hardware and the monitor will allow part of your 32K address space to reside in extended memory.

The PVAS is divided into eight sections called pages, numbered 0–7. Each page contains 4K words. RT–11 references each page by the Active Page Register (APR). The APR contains the relocation constant, which controls the mapping for each page. Figure 11–8 illustrates the PVAS, divided into pages. Keep in mind the structure of your program in terms of how it uses the virtual address space so that you can design its overlay structure correctly and efficiently.

Figure 11–8: Program Virtual Address Space



Each overlay that is to reside in extended memory must start on one of the 4K-word pge boundaries. The linker automatically rounds up the size of each segment to achieve this. The linker thereby restricts you to a region reserved for the root, and a maximum of seven virtual overlay regions, each starting on a page boundary. If any of these segments extends beyond a 4K word boundary, then one fewer virtual overlay regions is available. For example, if the root is 5K words long, then the static region uses the addresses referenced by APRs 0 and 1. Only six virtual overlay regions will remain, those referenced by APRs 2 through 7.

11.5.2.2 Physical Address Space – When LINK creates the load module for a program that has overlays in extended memory, it defines how each overlay will be mapped to extended memory during run time. LINK handles extended memory overlays differently from low memory overlays. Figures 11–9 and 11–10 compare the differences.

Figure 11–9 shows the physical address space of a program that has low memory overlays. Overlay segments share each region, and each is read in from an auxiliary mass storage volume when called.

Figure 11–9: Physical Address Space for Program with Low Memory Overlays

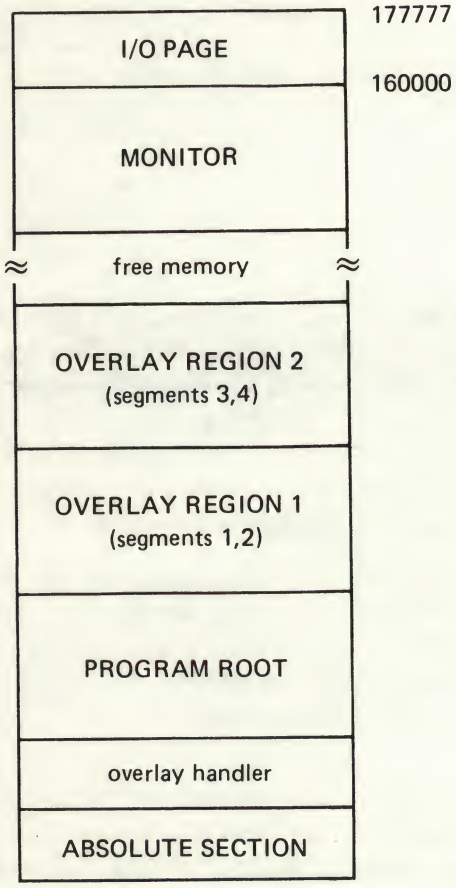
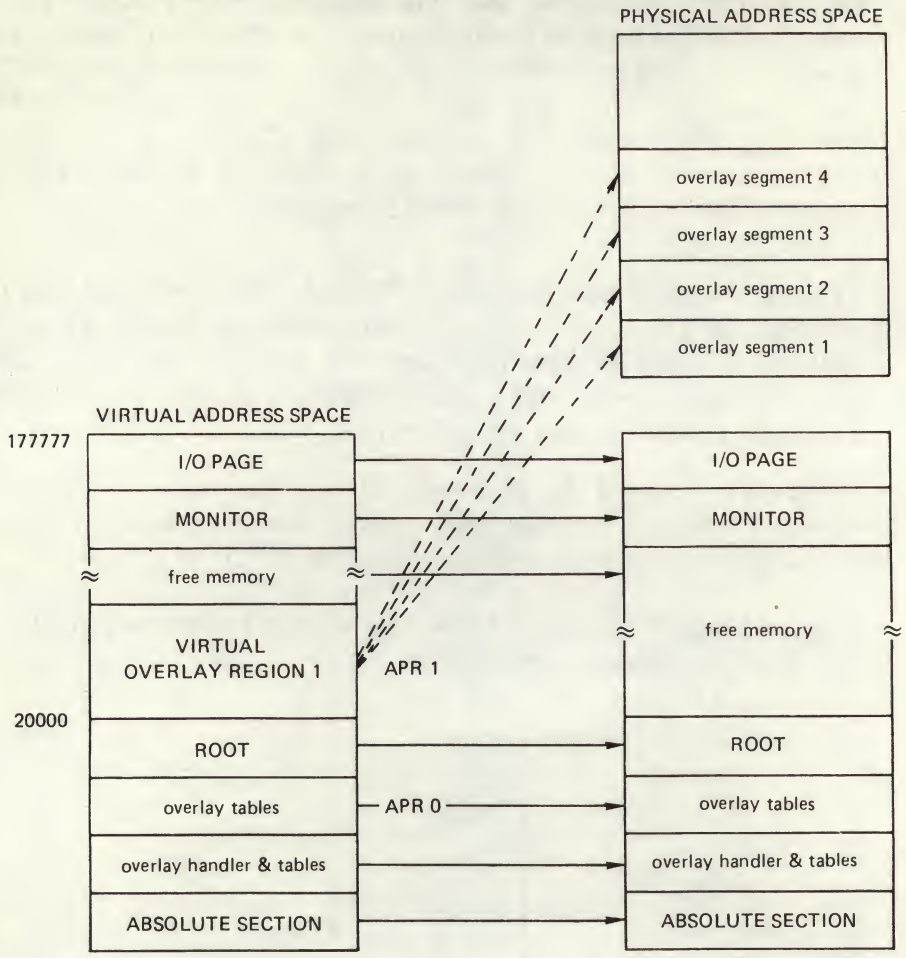


Figure 11–10: Virtual and Physical Address Space



In Figure 11–10, the diagram on the left shows the program virtual address space (0 to 177777). The diagram on the right shows the physical address space. In the program virtual address space, there is only one overlay region, and it starts on a 4K word boundary (APR 1 references this region). The regions of address space that will map to extended memory are called virtual overlay regions. Notice the arrows that point from the virtual overlay region to a number of overlay segments that appear on the right.

The overlay segments in the virtual overlay region shown use the space specified by APR 1 (20000 to 37777), but they occupy contiguous areas of extended memory, called partitions. At run time, overlay segments 1 through 4, once called, are concurrently resident in extended memory, and no further disk I/O is done to access these segments.

11.5.2.3 Virtual and Privileged Jobs – The amount of virtual address space available to your program depends on the type of program you are running. Background, foreground, and system jobs can fall into two categories: virtual and privileged.

Virtual jobs can use all 32K words of the virtual address space, but they cannot directly access the I/O page, the monitor, the vectors, or other jobs. Unless you need to access these protected areas of memory, make your jobs virtual by setting bit 10 of the JSW.

Privileged jobs also have 32K words of virtual addressing space, but by default, the protected areas (monitor, I/O page, vectors, and so on) are part of this addressing space. Just as you may lose access to protected areas if you implement your own extended memory mapping, you may lose access to the monitor and I/O page if you use extended memory overlays with a privileged job.

Virtual and privileged jobs can map to extended memory. You can use extended memory overlays with any type of virtual or privileged job (foreground, system, background).

See the *RT–11 Software Support Manual* for more details on virtual and privileged jobs, and see the *RT–11 Programmer's Reference Manual* for instructions on how to make a job virtual.

11.5.2.4 Extended Memory Overlay Option (/V:n[:m]) – Use the /V option to describe your program's structure in terms of virtual overlay regions (areas of virtual address space) and partitions (areas of physical address space). The argument, n, represents a virtual overlay region, and m represents a partition. As you specify successive extended memory overlay segments in the command string, make sure that the n and m in the /V:n[:m] notation are in ascending order. The following examples show how to use the /V:n[:m] option.

In the first example, program PROG has four segments to be mapped to extended memory. The four segments are named SEG1, SEG2, SEG3, and SEG4.

```
.R LINK
*PROG=PROG//
*SEG1/V:1
*SEG2/V:1
*SEG3/V:1
*SEG4/V:1//
```

These segments map to extended memory exactly as Figure 11–10 shows. Notice how each segment fits into its own partition in extended memory. Because each segment fits into its own partition, no storage volume access is necessary to change (or swap) segments once they have been read in.

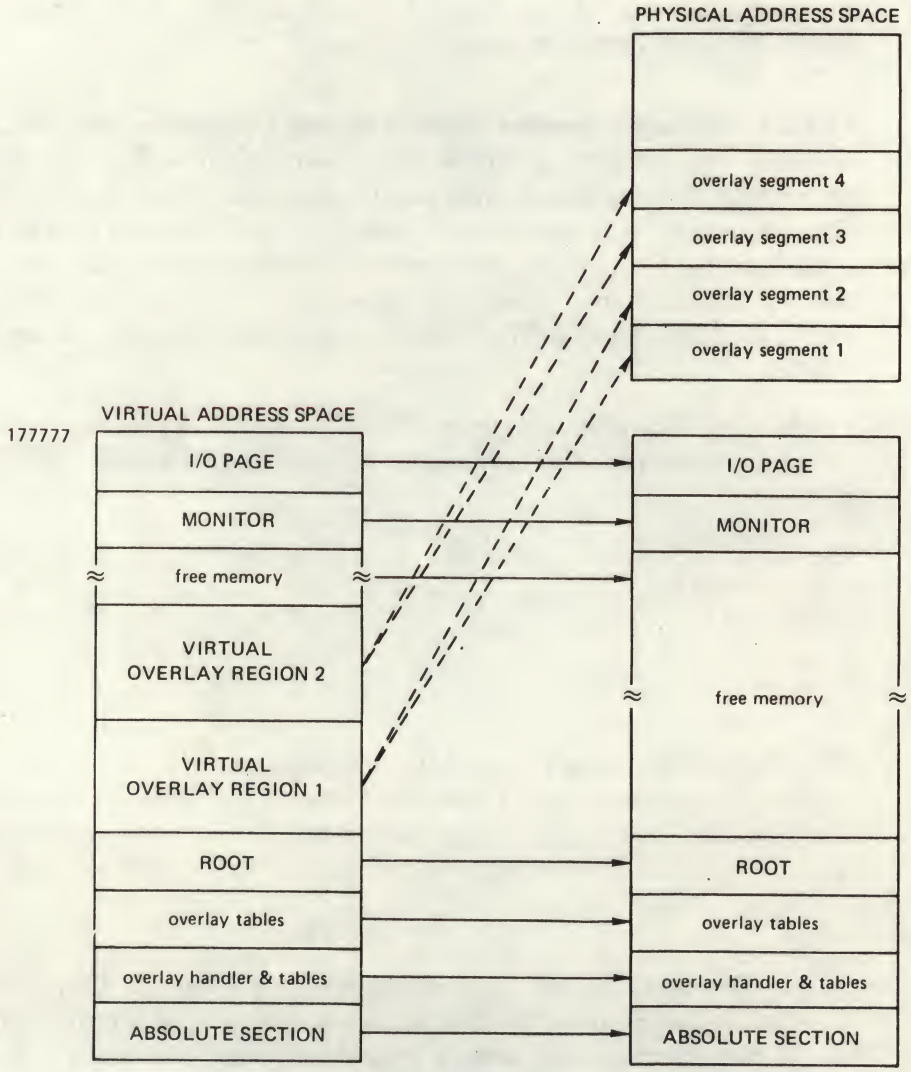
NOTE

The /V:n[:m] option works differently from the /O:n option. If /O:n were used in the previous example, the four segments would share the same physical locations, obviously requiring storage volume I/O as each segment is called. With /V:n[:m], each segment from the previous example occupies a unique area in extended memory, and no mass storage I/O is necessary after each segment is called.

The next example places the same four segments into virtual overlay regions 1 and 2. Although the program in this example uses two virtual overlay regions at run time, the segments will reside in memory the same as the segments shown in Figure 11–10. The virtual address space will be different for this example, however (see Figure 11–11). SEG1 and SEG2 use APR 1 (20000 to 37777), while SEG3 and SEG4 use APR 2 (40000 to 57777).

```
. R LINK
* PROG=PROG//
* SEG1/V:1
* SEG2/V:1
* SEG3/V:2
* SEG4/V:2//
```

Figure 11–11: Virtual and Physical Address Space

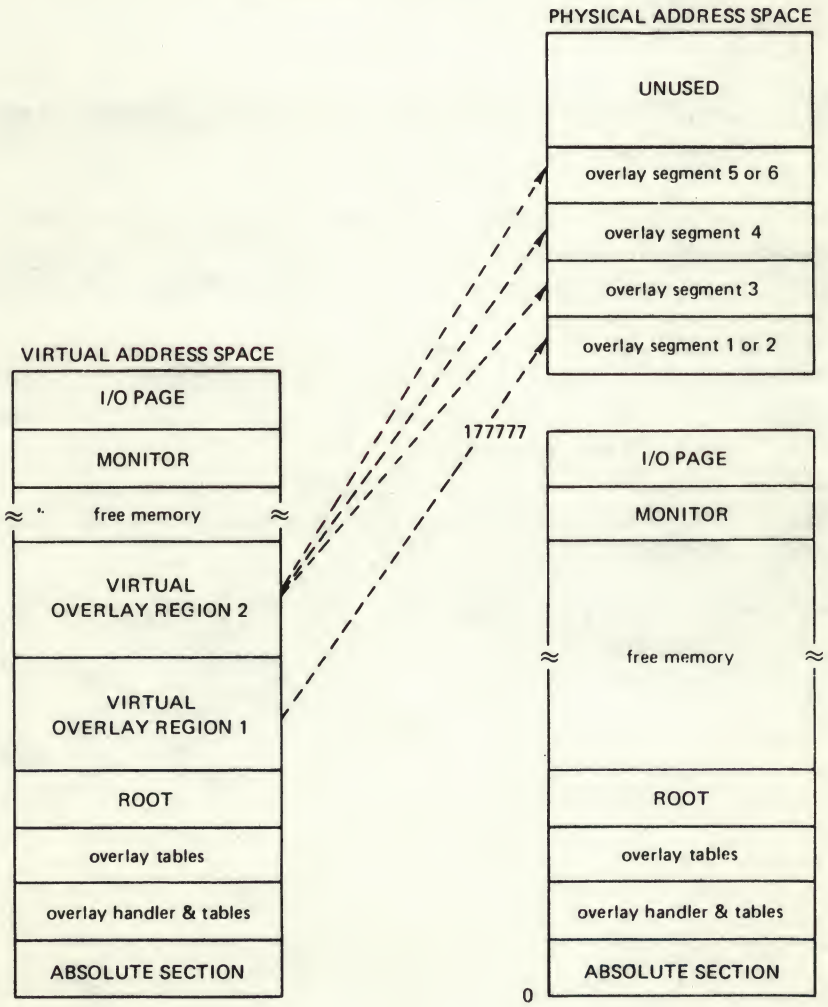


The argument, m in /V:n[:m], represents the partition in extended memory for the overlay segment. If you use m, segments can share the same partition in extended memory. That is, a segment, when called by your program, can be read in from auxiliary storage, thus overlaying the segment that currently occupies the same partition. When segments share partitions, the program requires auxiliary storage for I/O during run time, as does a program with low memory overlays.

LINK makes each partition the size of the largest segment it must accommodate. The following example generates the overlay structure shown in Figure 11–12.

```
. R LINK
* PROG=PROG//
* SEG1/V:1:1
* SEG2/V:1:1
* SEG3/V:2
* SEG4/V:2
* SEG5/V:2:1
* SEG6/V:2:1//
```

Figure 11–12: Extended Memory Partitions that Contain Sharing Segments



Notice that there are four segments specified for virtual overlay region 2, and that two segments share partition 1. The m value in /V:n:m groups segments in a region. The only reason to use the argument m is to create a partition that contains two or more segments. As shown in the previous example, the m argument is specified in ascending order within each virtual overlay region. This means you can renumber m from 1 for each virtual overlay region.

If you specify four segments for the same virtual overlay region, as in Example 1 below, the result is the same as if you specified Example 2. Because two segments are not specified to share the same partition, the partition order is as Example 2 shows.

Example 1

```
* SEG1/V:1
* SEG2/V:1
* SEG3/V:1
* SEG4/V:1
```

Example 2

```
* SEG1/V:1:1
* SEG2/V:1:2
* SEG3/V:1:3
* SEG4/V:1:4
```

11.5.3 Combining Low Memory Overlays with Extended Memory Overlays

You can combine low memory overlays and extended memory overlays in the same program structure. If you do so, however, each low memory overlay region you use makes your remaining virtual address space smaller.

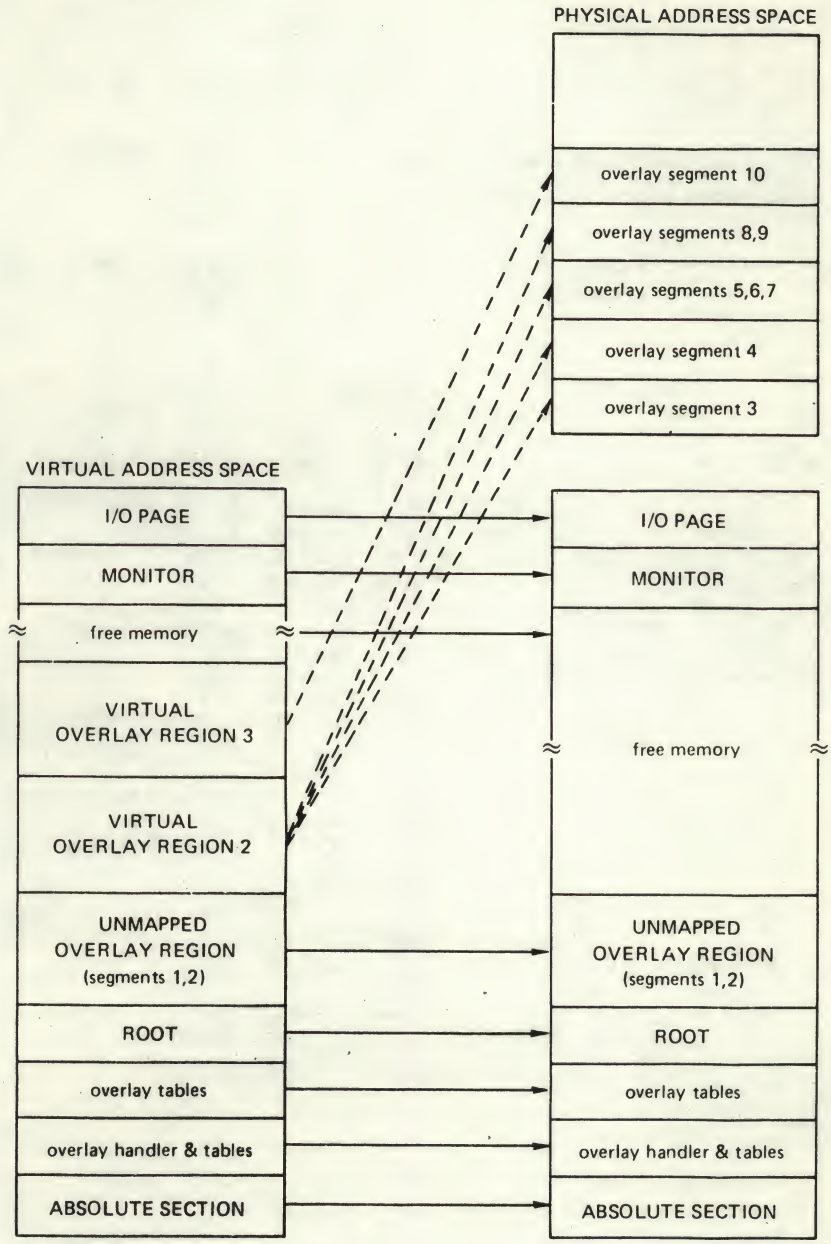
It is important to note that as you combine low memory overlays with extended memory overlays, you must list your regions in ascending order, whether or not one is a low memory overlay region and the next is a virtual region. That is, if the first overlay region is a low memory overlay region, specify it as region 1. If the next region is a virtual region, specify it as region 2. Note that you must specify low memory overlays before extended memory overlays.

The following example creates a low memory overlay region and a virtual overlay region above it.

```
* R LINK
* PROG=PROG//
* SEG1/O:1
* SEG2/O:1
* SEG3/V:2
* SEG4/V:2
* SEG5/V:2:1
* SEG6/V:2:1
* SEG7/V:2:1
* SEG8/V:2:2
* SEG9/V:2:2
* SEG10/V:3//
```

Figure 11–13 shows how low memory and extended memory might appear if the program from this example were loaded.

Figure 11–13: Memory Diagram Showing Low Memory and Extended Memory Overlays



11.5.4 Load Map

Figure 11–14 shows a sample load map for PROG.SAV, whose overlay structure is defined below.

```
* PROG,PROG=MOD0//
* MOD1/O:1
* MOD2/O:1
* MOD3/V:2
* MOD4/V:3//
```

Table 11–9 describes the portions of this load map devoted to low memory and extended memory overlays.

Figure 11–14: Load Map for Program with Unmapped and Virtual Overlays

```
1    RT-11 LINK  V08.00       Load Map        Thursday 04-Nov-82 14:15  Page 1
2    V     .SAV       Title:  .MAIN.  Ident:
3
4    Section  Addr   Size    Global  Value   Global  Value   Global  Value
5
6     . ABS.  000000 001000 = 256.   words  (RW,I,GBL,ABS,OVR)
7     $OHAND  001000 000252 = 85.    words  (RW,I,GBL,REL,CON)
8                            $OVRHV  001000  $OVRH   001004  V$READ  001034
9                            V$DONE  001046  $VDF5   001234  $VDF4  001236
10                           $VDF1   001246  $VDF2   001250
11    $OTABL  001252 000114 = 38.    words  (RW,D,GBL,REL,OVR)
12            001366 000410 = 132.   words  (RW,I,LCL,REL,CON)
13    MAIN    001776 000070 = 28.    words  (RW,I,LCL,REL,CON)
14                           START   001776 RET1     002010  RET2    002014
15                           LIMIT   002024
16    LML4    002066 000026 = 11.    words  (RW,I,GBL,REL,CON)
17                           MSGL    002066
18    LML5    002114 000026 = 11.    words  (RW,I,GBL,REL,CON)
19                           MSGL2   002114
20    Segment size = 002142 = 561.   words
21
22   Overlay  region 000001  Segment 000001
23    LML2    002144 000032 = 13.    words  (RW,I,LCL,REL,CON)
24                           START1@ 002144
25    Segment size = 000032 = 13.    words
26
27   Overlay  region 000001  Segment 000002
28    LML3    002144 000036 = 15.    words  (RW,I,LCL,REL,CON)
29                           START2@ 002144
30    Segment size = 000036 = 15.    words
31
32   ----------------------------------------------------------------------
33
34    Virtual overlay region 000002
35    ----------------------
36
37    Partition      000001  Segment 000003
38     LML7   020002 000034 = 14.    words  (RW,I,LCL,REL,CON)
39                           START3  020002
40     LML6   020036 000042 = 17.    words  (RW,I,GBL,REL,CON)
41                           MSGL3   020036 RET4     020050
42     Segmentsize = 000076 = 31.    words
43
44    Virtual overlay region 000003
45    ----------------------
46
```

(Continued on next page)

```
47    Partition       000002  Segment 000004
48     LML9   040002 000076 = 31.    words  (RW,I,GBL,REL,CON)
49                           MSGL9@  040002
50     Segment size = 000076 = 31.   words
51
52
53    Transfer address = 001776, High limit = 002200 = 576.  words
54
55
56    Virtual high limit = 040076 = 8223. words, next free address = 060000
57
58
59    Extended memory required = 000200 = 64.        words
60    RT-11 LINK  V08.00  Global Symbol Cross Reference Table  Page 1
61
62
63    $OVDF1 VHANDL+
64    $OVDF2 VHANDL+
65    $OVDF3 VHANDL+
66    $OVDF4 VHANDL+
67    $OVDF5 VHANDL+
68    $OVRH  VHANDL#+
69    $OVRHV VHANDL#+
70    $VDF1  VHANDL#+
71    $VDF2  VHANDL#+
72    $VDF4  VHANDL#+
73    $VDF5  VHANDL#+
74    LIMIT  .MAIN.#
75    MSGL   .MAIN.#
76    MSGL2  .MAIN.#
77    MSGL3  .MAIN.#
78    MSGL9  .MAIN.    .MAIN.#
79    RET1   .MAIN.#
80    RET2   .MAIN.#
81    RET4   .MAIN.#
82    START  .MAIN.#
83    START1 .MAIN.    .MAIN.#
84    START2 .MAIN.    .MAIN.#
85    START3 .MAIN.#
86    V$DONE VHANDL#+
87    V$READ VHANDL#+
```

Table 11–9 gives a line-by-line description of the load map above. This table makes references only to those portions of the load map that are unique to overlaid programs, and also describes the global cross-reference table (which is not unique to overlaid programs). For details on other parts of the load map, see Section 11.4.4.

Table 11–9: Line-by-Line Sample Load Map Description

Line	Description
7–10	$OHAND p-sect. This is the overlay handler for overlays in both low and extended memory.
11	$OTABL p-sect. This program section contains tables of data used by the overlay handler.
12	Blank p-sect. The load map for overlaid programs lists the blank p-sect, when present, after the $OHAND and $OTABL p-sects.
20	Contains data about the size of the program's root. The sections of the load map that follow provide information on the part of the program that is overlaid.

(Continued on next page)

Table 11–9: Line-by-Line Sample Load Map Description (Cont.)

Line	Description
22	Header for overlay region 1, segment 1 (low memory overlay region).
23–24	LML2 p-sect. This is the only p-sect in segment 1. Notice in line 24 the @ character next to the global START1. This character indicates that its associated global is accessed through data contained in the overlay table p-sect, $OTABL, which is in the root.
25	Contains data on the size of segment 1.
32	Delineates the portion of the load map devoted to low memory from the portion devoted to extended memory.
34	Header for virtual overlay region 2. Note that overlay regions are numbered in ascending order, whether in low or extended memory.
37	Header for partition 1, segment 3.
41	Notice the absence of the @ character for the globals in p-sect LML6. This indicates that LML6 is not called outside segment 3.
42	Contains data on the size of overlay segment 3.
44	Header for virtual overlay region 3.
47	Header for partition 2, segment 4.
50	Contains data on the size of segment 4. Notice that segments 3 and 4 have the same length. LINK automatically rounds up the size of virtual overlay segments to multiples of 32 (decimal) words (or 100 octal bytes). LINK adds an overlay segment number word to the segment size number (the number 000076 that follows 040002 in line 48) to give the actual segment size.
53	Transfer address and high limit. The transfer address is the start address of the program. The high limit is the last low memory address used by the root and unmapped overlays.
56	Virtual high limit. Indicates the last virtual address used by the part of the program in extended memory. The next free address is the address of the next page not in use by the program.
59	Indicates the amount of extended memory required by the program. Make sure you check this figure to ensure you have adequate space for your program at run time.
60–87	Cross-reference section of defined global symbols. Displays a cross-reference of all global symbols defined during the linking process. Note that global symbols are listed alphabetically and are followed by the names of the modules in which the global symbols are either defined or referenced. A pound sign (#) following a module name indicates that the global symbol is defined in that module. A plus sign (+) following a module name indicates that the module is from a library.

Figure 11–15 shows the extended memory overlay handler.

Figure 11–15: Extended Memory Overlay Handler

```
.SBTTL  THE RUN-TIME OVERLAY HANDLER
.ENABL GBL

;+
; THE FOLLOWING CODE IS INCLUDED IN THE USER'S PROGRAM BY THE
; LINKER WHENEVER LOW MEMORY OVERLAYS ARE REQUESTED BY THE USER.
; THE RUN-TIME LOW MEMORY OVERLAY HANDLER IS CALLED BY A DUMMY
; SUBROUTINE OF THE FOLLOWING FORM:
;
;       JSR     R5,$OVRH        ;CALL TO COMMON CODE FOR LOW MEMORY OVERLAYS
;       .WORD   <OVERLAY #*6>   ;# OF DESIRED SEGMENT
;       .WORD   <ENTRY ADDR>    ;ACTUAL CORE ADDRESS (VIRTUAL ADDRESS)
;
; ONE DUMMY ROUTINE OF THE ABOVE FORM IS STORED IN THE RESIDENT PORTION
; OF THE USER'S PROGRAM FOR EACH ENTRY POINT TO A LOW MEMORY OVERLAY SEGMENT.
; ALL REFERENCES TO THE ENTRY POINT ARE MODIFIED BY THE LINKER TO BE
; REFERENCES TO THE APPROPRIATE DUMMY ROUTINE. EACH OVERLAY SEGMENT
; IS CALLED INTO CORE AS A UNIT AND MUST BE CONTIGUOUS IN CORE. AN
; OVERLAY SEGMENT MAY HAVE ANY NUMBER OF ENTRY POINTS, TO THE LIMITS
; OF CORE MEMORY. ONLY ONE SEGMENT AT A TIME MAY OCCUPY AN OVERLAY REGION.
;
; IF OVERLAYS IN EXTENDED MEMORY ARE SPECIFIED, THE FOLLOWING DUMMY SUBROUTINE
; IS USED AS THE ENTRY POINT TO THE EXTENDED MEMORY OVERLAY HANDLER.
;
;       JSR     R5,$OVRHV       ;ENTRY FOR /V (EXTENDED MEMORY) OVERLAYS
;       .WORD   <OVERLAY #*6>   ;# OF DESIRED SEGMENT
;       .WORD   <VIRTUAL ENTRY ADDRESS> ;VIRTUAL ADDRESS OF SEGMENT
;
; ADDITIONAL DATA STRUCTURES IN THE EXTENDED MEMORY OVERLAY HANDLER AND THE
; OVERLAY TABLE PERMIT USE OF EXTENDED MEMORY. ONE REGION DEFINITION
; BLOCK IS DEFINED IN THE HANDLER, AND XM EMT'S ARE ALSO INCLUDED. WINDOW
; DEFINITION BLOCKS FOR THE EXTENDED MEMORY PARTITIONS FOLLOW THE DUMMY
; SUBROUTINES IN THE OVERLAY TABLE.
;
; THERE IS ONE WORD PREFIXED TO EVERY OVERLAY REGION THAT IDENTIFIES THE
; SEGMENT CURRENTLY RESIDENT IN THAT OVERLAY REGION. THIS WORD IS AN INDEX
; INTO THE OVERLAY TABLE AND POINTS AT THE OVERLAY SEGMENT INFORMATION.
;
; UNDEFINED GLOBALS IN THE OVERLAY HANDLER MUST BE NAMED "$OVDF1" TO
; "$OVDFn" SUCH THAT A RANGE CHECK MAY BE DONE BY LINK TO DETERMINE IF
; THE UNDEFINED GLOBAL NAME IS FROM THE OVERLAY HANDLER. A CHECK IS
; DONE ON THE .RAD50 CHARACTERS "$OV", AND THEN A RANGE CHECK IS DONE ON
. ; THE .RAD50 CHARACTERS "DF1" TO "DFn". THESE GLOBAL SYMBOLS DO NOT APPEAR
; ON LINK MAPS, SINCE THEIR VALUE IS NOT KNOWN UNTIL AFTER THE MAP HAS BEEN
; PRINTED. CURRENTLY $OVDF1 TO $OVDF5 ARE IN USE.
;
; GLOBAL SYMBOLS V$READ AND V$DONE ARE USEFUL WHEN DEBUGGING OVERLAID
; PROGRAMS.
;
;       V$READ:: WILL APPEAR IN THE LINK MAP AND LOCATES THE .READW
;       STATEMENT IN THE OVERLAY HANDLER.
;
;       V$DONE:: WILL APPEAR IN THE LINK MAP AND LOCATES THE FIRST
;       INSTRUCTION AFTER THE .READW IN THE OVERLAY HANDLER.
;-

.MCALL  .WDBDF,.RDBDF,.PRINT,.EXIT,.READW,..V1..
        ..V1..                  ;V1 FORMAT
        .WDBDF                  ;DEFINE WDB OFFSETS
        .RDBDF                  ;DEFINE RDB OFFSETS

.SBTTL  OVERLAY HANDLER CODE

.PSECT  $OHAND,GBL

.ENABL  LSB
```

(Continued on next page)

```
;+
; THERE ARE TWO ENTRY POINTS TO THE OVERLAY HANDLER: $OVRHV FOR /V
; (EXTENDED MEMORY) OVERLAYS, AND $OVRH FOR /O (LOW MEMORY) OVERLAYS.
;-

$OVRHV::INC     (PC)+           ;SET /V OVERLAY ENTRY SWITCH
10$:    .WORD   0               ;=0 IF /O ; =1 IF /V OVERLAY ENTRY
$OVRH:: MOV     R0,-(SP)        ;/O OVERLAY ENTRY POINT
        MOV     R1,-(SP)        ;SAVE REGISTERS
        MOV     R2,-(SP)

20$:
        BR      90$             ;FIRST CALL ONLY * * *
;       MOV     @R5,R1          ;PICK UP OVERLAY NUMBER
        ADD     #$OVTAB-6,R1    ;CALCULATE TABLE ADDRESS
        MOV     (R1)+,R2        ;GET FIRST ARG. OF OVERLAY SEG. ENTRY
        TST     10$             ;IS THIS /V ENTRY?
        BNE     60$             ;IF NON-ZERO THEN YES
30$:    CMP     (R5)+,@R2       ;IS OVERLAY ALREADY RESIDENT?
        BEQ     40$             ;YES, BRANCH TO IT

;+
; THE .READW ARGUMENTS ARE AS FOLLOWS:
; CHANNEL NUMBER, CORE ADDRESS, LENGTH TO READ, RELATIVE BLOCK ON DISK.
; THESE ARE USED IN REVERSE ORDER FROM THAT SPECIFIED IN THE CALL.
;-

V$READ::.READW  17,R2,@R1,(R1)+ ;READ FROM OVERLAY FILE
V$DONE::BCS     50$
40$:    MOV     (SP)+,R2        ;RESTORE USERS REGISTERS
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        MOV     @R5,R5          ;GET ENTRY ADDRESS
        CLR     10$             ;CLEAR /V FLAG
        RTS     R5              ;ENTER OVERLAY ROUTINE AND RESTORE USER'S R5

50$:    EMT     376             ;SYSTEM ERROR 10 (OVERLAY I/O)
        .BYTE   0,373
;+
; VIRTUAL OVERLAY SEGMENTS IN THE SAME REGION BUT IN DIFFERENT PARTITIONS
; USE DIFFERENT WDB'S.  ONLY ONE OF THESE WINDOWS EXISTS AT ANY TIME.
; THIS IS BECAUSE WHEN A NEW WINDOW IN A VIRTUAL OVERLAY REGION IS CREATED,
; THE MONITOR IMPLICITLY ELIMINATES ANY WINDOW THAT EXISTS IN THAT
; VIRTUAL OVERLAY REGION.  THUS, IF THE CALLED OVERLAY SEGMENT IS NOT
; CURRENTLY MAPPED, ITS WINDOW MUST BE RE-CREATED (.CRAW'ED) BESIDES
; BEING MAPPED.  THE MAPPING IS DONE IMPLICITLY IN THE FOLLOWING CODE
; SINCE THE WS.MAP BIT IS SET IN ALL OF THE VIRTUAL OVERLAY SEGMENTS'
; WDB'S.
;-
60$:    TSTB    @R2             ;DO WE NEED TO CREATE A WINDOW (.CRAW)?
        BEQ     70$             ;YES
        MOV     @W.NBAS(R2),R0  ;GET INDEX OF SEGMENT NOW MAPPED
        BEQ     70$             ;THERE ISN'T ONE; WE MUST .CRAW
        CMP     $OVTAB-6(R0),R2 ;IS OVERLAY REGION SAME AS THIS ONE?
        BEQ     80$             ;IF EQUAL, JUST WORRY ABOUT DISK I/O
70$:    MOV     #AREA+2,R0      ;POINT TO EMT ARGUMENT BLOCK + 2
        MOV     R2,@R0          ;STUFF ADDRESS OF WDB IN EMT AREA
        MOV     30.,*^O400+2.,-(R0) ;STUFF .CRAW CODE; R0 ->EMT AREA
        EMT     375             ;DO THE EMT
        BCS     110$            ;CARRY SET MEANS ERROR!
80$:    MOV     W.NBAS(R2),R2   ;GET MEMORY ADDRESS OF OVERLAY
        BR      30$             ;CHECK IF DISK I/O IS NECESSARY

90$:    MOV     #11501,20$      ;RESTORE SWITCH INSTR (MOV @R5,R1)
        MOV     $VDF1,R1        ;START ADDRESS FOR CLEAR OPERATION
100$:   CMP     R1,$VDF2        ;ARE WE DONE?
        BHIS    20$             ;HIS -> DONE, OR NO /O OVERLAYS
        CLR     (R1)+           ;CLEAR ALL LOW MEMORY OVERLAY REGIONS
        BR      100$
```

(Continued on next page)

```
; ERROR MESSAGE

110$:   MOV     #MSG2,R0        ;OTHERWISE ERROR
        .PRINT                  ;AND PRINT MESSAGE
        .EXIT                   ;AND EXIT

.DSABL  LSB

.SBTTL IMPURE AREA

.ENABL  LC
.NLIST BEX
MSG2:   .ASCIZ  /VHANDL-F-Window error/
LIST BEX

.EVEN
AREA:   .WORD   0,0             ;EMT AREA BLOCK FOR .CRAW

$VDF5:: .WORD   $OVDF5          ;POINTER TO WORD AFTER WDB'S IN OVERLAY TABLE
$VDF4:: .WORD   $OVDF4          ;POINTER TO START OF WDB'S IN OVERLAY TABLE

RGADR:  .WORD   0               ;THREE WORD REGION DEFINITION BLOCK
RGSIZ:  .WORD   $OVDF3,0        ;$OVDF3 -> SET BY LINK = SIZE OF REGION

$VDF1:: .WORD   $OVDF1          ;HIGH ADDR ROOT SEGMENT + 2 (NXT AVAIL)
$VDF2:: .WORD   $OVDF2          ;HIGH ADDR /O OVERLAYS + 2 (NXT AVAIL)

.SBTTL  $OVTAB  OVERLAY TABLE

;+
; OVERLAY TABLE STRUCTURE:
;
; LOC 64 ->     $OVTAB:
;               .WORD   <CORE ADDR>,<RELATIVE BLK>,<WORD COUNT>   /O OVERLAYS
; LOC 66 ->     .WORD   <WDB ADDR>,<RELATIVE BLK>,<WORD COUNT>    /V OVERLAYS
;               DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
; $VDF4 ->      WINDOW DEFINITION BLOCKS FOR EXTENDED MEMORY OVERLAYS (/V)
; $VDF5 ->      WORD AFTER THE END OF THE WINDOW DEFINITION BLOCKS (/V)
;-

.PSECT  $OTABL,D,GBL,OVR

$OVTAB:

.END
```

11.6 Options

Full descriptions of the options summarized in Table 11–6 follow in alphabetical order.

11.6.1 Alphabetical Option (/A)

The /A option lists global symbols in program sections in alphabetical order.

11.6.2 Bottom Address Option (/B:n)

The /B:n option supplies the lowest address to be used by the relocatable code in the load module. The argument, n, is a six-digit unsigned, even octal number that defines the bottom address of the program being linked. If you do not supply a value for n, the linker prints:

```
?LINK-F-/B no value
```

Retype the command line, supplying an even octal value.

When you do not specify /B, the linker positions the load module so that the lowest address is location 1000 (octal). If the ASECT size is greater than 1000, the size of ASECT is used.

If you supply more than one /B option during the creation of a load module, the linker uses the first /B option specification. /B is invalid when you are linking to a high address (/H). The /B option is also invalid with foreground links. Foreground modules are always linked to a bottom address of 1000 (octal).

The bottom value must be an unsigned, even, octal number. If the value is odd, the ?LINK-F-/B odd value error message prints. Reenter the command string specifying an unsigned, even octal number as the argument to the /B option.

11.6.3 Continuation Option (/C or //)

The continuation option (/C or //) lets you type additional lines of command string input.

Use the /C option at the end of the current line and repeat it on subsequent command lines as often as necessary to specify all the input modules in your program. Do not enter a /C option on the last line of input.

The following command indicates that input is to be continued on the next line; the linker prints an asterisk.

```
* OUTPUT,LP:=INPUT/C
*
```

An alternate way to enter additional lines of input is to use the // option on the first line. The linker continues to accept lines of input until it encounters another // option, which can be either on a line with input file specifications or on a line by itself. The advantage of using the // option instead of the /C option is that you do not have to type the // option on each continuation line. This example shows the command file that links the linker:

```
R LINK
LINK,LINK=LINK0,LNKLB1/D//
LINK1/O:1
LINK2/O:1
LINK3/O:1
LINK4/O:1
LINK5/O:1
LINK6/O:1
LINK7/O:1
LINK8/O:1
LNKEM/O:1//
BITST
GETBUF
WRITO
WRTLRU
ZSWFIL
(RET)
```

You cannot use the /C option and the // option together in a link command sequence. That is, if you use // on the first line, you must use // to terminate input on the last line. If you use /C on the first line, use /C on all lines but the last.

11.6.4 Duplicate Global Symbol Option (/D)

The /D option allows you to specify library modules that you want to reside in more than one overlay segment. Type /D on the first command line. After you have typed all input command lines, the linker prompts:

```
Duplicate symbol?
```

Type the names of the global symbols in the library module that you want to be defined once in each segment that references those symbols. Follow each global symbol with a carriage return. A carriage return on a line by itself terminates the list of symbols. Only global symbols defined in library modules can be duplicated. If you use the /D option and specify a global symbol that is defined outside of a library module, the symbol definition is not duplicated and LINK prints the message ?LINK-W-Duplicate symbol SYMBOL defined in DEV:FILNAM.TYP.

When you do not use the /D option and a global symbol defined in a library module is externally referenced (that is, the global symbol is referenced from a segment other than the one in which it is defined), the linker places the library module in the program's root segment. Therefore, if a library module is referenced by more than one global symbol, each of the global symbols in the library module that is referenced should be named in response to the /D option. Otherwise, the library module will be placed in the root segment. Also, if any of a library module's global symbols are referenced from the root, the library module will be placed in the root even if you have named the global symbols in response to the /D option. In each of these cases when a library module that link places in the root contains global symbols declared with /D, LINK prints the following message and the global symbol is not duplicated.

```
?LINK-W-Duplicate symbol SYMBOL is forced to the root
```

Special Programming Considerations for the /D Option

Even when a library module you duplicate is not referenced from the root, any global section within that module that is referenced from more than one segment is always placed in the root. If local sections within the same library module have no need to communicate with each other, define the global section with the CON attribute. This causes the linker to place a separate copy of the global section in the root for each copy of the library module's local sections placed in overlays. Although the global section resides in the root while the local sections reside in overlays, each copy of the library module retains its identity as a separate copy of the module. Since each copy of the global section is bound to its own local section in an overlay, this ensures that references between the local and global sections will be bound to the correct definitions.

However, when a library module that you want to duplicate will be placed in overlay segments that exchange information, another consideration exists. If the library module contains a section of global data to be referenced by local sections within the module, but the global section does not reference any local section within the module, you should move a copy of the global section to the root. To move this section to the root, define the section with a unique name and give the section the GBL and OVR attributes. When this section is placed in the root, the local sections from the duplicated library module that reside in the overlay segments can reference the global section in the root. Since the global section has been given the OVR attribute rather than CON, the local sections can pass information to specific locations in the global section, and the local sections can access the same locations to send and receive data.

Figure 11–16 illustrates a duplicated library module whose global data section has been forced to the root with the CON attribute. The arrows show each local section accessing information from its copy of the global section within the root. Notice, however, that the local sections (which are identical) cannot exchange data becuse their references are bound to different locations. Figure 11–17 illustrates the same duplicated library module, this time with the global data section forced to the root with the OVR attribute. Notice that the two local sections can now reference the same location in the global section to exchange information.

Figure 11–16: Global Data Section with CON Attribute

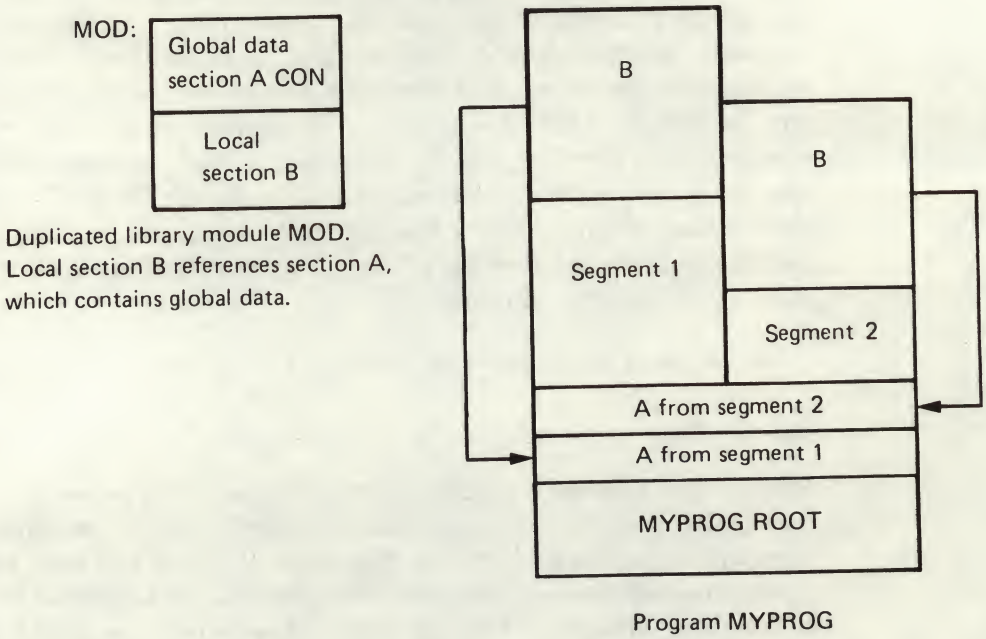
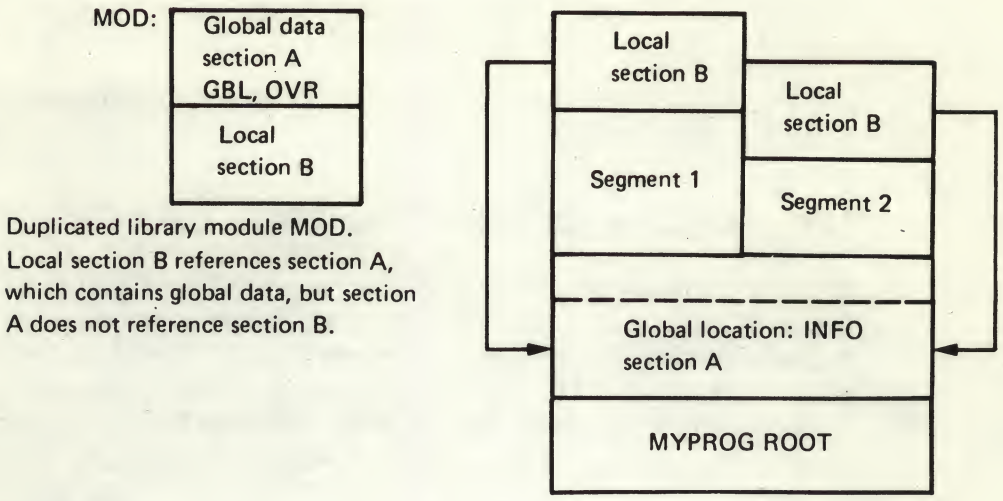


Figure 11–17: Global Data Section with OVR Attribute



11.6.5 Extend Program Section Option (/E:n)

The /E:n option allows you to extend a program section in the root to a specific value. Type the /E:n option at the end of the first command line. After you have typed all input command lines, the linker prompts with:

```
Extend section?
```

Respond with the name of the program section to be extended, followed by a carriage return. The resultant program section size is equal to or greater than the value you specify, depending on the space the object code requires. The value you specify must be an even byte value. Note that you can extend only one section.

The following example extends section CODE to 100 (octal) bytes.

```
* X,TT:=LK001/E:100
Extend section? CODE
```

11.6.6 Default FORTRAN Library Option (/F)

By indicating the /F option in the command line, you can link the FORTRAN library (FORLIB.OBJ on the system device SY:) with the other object modules you specify. You do not need to specify FORLIB explicitly. For example:

```
* FILE,LP:=AB/F
```

The object module AB.OBJ from DK: and the required routines from the FORTRAN library SY:FORLIB.OBJ are linked together to form a load module called FILE.SAV.

The linker automatically searches a default system library, SY:SYSLIB.OBJ. The library normally includes the modules that compose FORLIB. The /F option is provided only for compatibility with other versions of RT–11. You should not have to use /F.

See the *RT–11 Installation Guide* for details on combining SYSLIB and FORLIB library files.

11.6.7 Directory Buffer Size Option (/G)

When you are using modules for your program that are from a multiple definition library, LINK has to store that library's directory in an internal buffer. Occasionally, this buffer area is too small to contain an entire directory, in which case LINK is unable to process those modules. The /G option instructs LINK to adjust the size of its directory buffer to accommodate the largest directory size of the multiple definition libraries you are using.

You should use /G only when required because it slows down linking time. Use it only after an attempt to link your program failed because the buffer was too small. When a link failure of this sort occurs, LINK prints the message ?LINK-F-Library EPT too big, increase buffer with /G.

11.6.8 Highest Address Option (/H:n)

The /H:n option allows you to specify the top (highest) address to be used by the relocatable code in the load module. The argument n represents an unsigned, even octal number. If you do not specify n, the linker prints:

```
?LINK-F-/H no value
```

Retype the command, supplying an even octal number to be used as the value.

If you specify an odd value, the linker responds with:

```
?LINK-F-/H odd value
```

Retype the command, supplying an even octal number.

If the value is not large enough to accommodate the relocatable code, the linker prints:

```
?LINK-F-/H value too low
```

Relink the program with a larger value.

The /H option cannot be used with the /R, /Y, or /B options.

NOTE

Be careful when you use the /H option. Most RT–11 programs use the free memory above the relocatable code as a dynamic working area for I/O buffers, device handlers, symbol tables, etc. The size of this area differs according to the memory configuration. Programs linked to a specific high address might not run in a system with less physical memory because there is less free memory.

11.6.9 Include Option (/I)

The /I option lets you take global symbols from any library and include them in the linking process even when they are not needed to resolve globals. This provides a method for forcing modules that are not called by other modules to be loaded from the library. All modules that you specify with /I go into the root. When you specify the /I option, the linker prints:

```
Library search?
```

Reply with the list of global symbols to be included in the load module; type a carriage return to enter each symbol in the list. A carriage return alone terminates the list of symbols.

The following example includes the global $SHORT in the load module:

```
* SCCA=RK1:SCCA/I (RET)
Library search?  $SHORT (RET)
Library search?  (RET)
```

11.6.10 Memory Size Option (/K:n)

The /K:n option lets you insert a value into word 56 of block 0 of the image file. The argument n represents the number of 1K words of memory required by the program; n is an integer in the range 2–28 (decimal). This option allows you to limit the amount of memory allocated by a .SETTOP request to nK words. You cannot use the /K option with the /R option.

11.6.11 LDA Format Option (/L)

The /L option produces an output file in LDA format instead of memory image format. The LDA format file can be output to any device including those that are not block-replaceable. It is useful for files that are to be loaded with the absolute loader. The default file type .LDA is assigned when you use the /L option. You cannot use the /L option with the low memory overlay option (/O), the foreground link option (/R), or the extended memory overlay option (/V).

The following example links files IN and IN2 on device DK: and outputs an LDA format file, OUT.LDA, to the diskette and a load map to the line printer.

```
* DY:OUT,LP:=IN,IN2/L
```

11.6.12 Modify Stack Address Option (/M[:n])

The stack address, location 42, is the address that contains the initial value for the stack pointer. The /M option lets you specify the stack address. If you use the /R:n option (foreground link) with /M, LINK ignores the value on /R:n. The argument n is an even, unsigned, six-digit octal number that defines the stack address.

After all input lines have been typed, the linker prints the following message if you have not specified a value for n:

```
Stack symbol?
```

In this case, specify the global symbol whose value is the stack address and follow with a carriage return. You must not specify a number. If you specify a nonexistent symbol, an error message prints and the stack address is set to the system default (1000 for .SAV files) or to the bottom address if you used /B. If the program's absolute section extends beyond location 1000, the default stack space starts after the largest .ASECT allocation of memory.

Direct assignment (with .ASECT) of the stack address within the program takes precedence over assignment with the /M option. The statements to do this in a MACRO program are as follows:

```
.ASECT
.=42
.WORD   INITSP      ;INITIAL STACK SYMBOL VALUE
.PSECT              ;RETURN TO PREVIOUS SECTION
```

The following example modifies the stack address.

```
* OUTPUT=INPUT/M (RET)
Stack symbol?   BEG (RET)
```

11.6.13 Cross-Reference Option (/N)

The /N option includes in the load map a cross-reference of all global symbols defined during the linking process. The global symbols are listed alphabetically. Each global symbol is followed by the names of the modules (also listed alphabetically) in which the symbol is defined or referenced. A pound sign (#) next to the module name indicates that the symbol is defined in that module. A plus sign (+) indicates that the module is from a library. The cross-reference section, if requested, begins on a new page at the end of the load map. See Figure 11–14 (and Table 11–9) for an illustration of a global cross-reference listing.

11.6.14 Low Memory Overlay Option (/O:n)

The /O option segments the load module so that the entire program is not memory resident at one time. This lets you execute programs that are larger than the available memory.

The argument n is an unsigned octal number (up to five digits in length) specifying the overlay region to which the module is assigned. The /O option must follow (on the same line) the specification of the object modules to which it applies, and only one overlay region can be specified on a command line. Overlay regions cannot be specified on the first command line; that is reserved for the root segment. You must use /C or // for continuation.

You specify coresident overlay routines (a group of subroutines that occupy the overlay region and segment at the same time) as follows:

```
* OBJA,OBJB,OBJC/O:1/C
* OBJD,OBJE/O:2/C
```

All modules that the linker encounters until the next /O option will be coresident overlay routines. If you specify, at a later time, the /O option with the same value you used previously (same overlay region), then the linker opens up the corresponding overlay area for a new group of subroutines. This group occupies the same locations in memory as the first group, but it is never needed at the same time as the previous group.

The following commands to the linker make R and S occupy the same memory as T (but at different times):

```
* MAIN,LP:=ROOT/C
* R,S/O:1/C
* T/O:1
```

The following example establishes two overlay regions.

```
* OUTPUT,LP:=INPUT//
* OBJA/O:1
* OBJB/O:1
* OBJC/O:2
* OBJD/O:2
* //
```

You must specify overlays in ascending order by region number. For example:

```
* A=A/C
* B/O:1/C
* C/O:1/C
* D/O:1/C
* G/O:2
```

The following overlay specification is invalid since the overlay regions are not given in ascending numerical order. An error message prints in each case, and the overlay option immediately preceding the message is ignored.

```
* X=LIBR0//
* LIBR1/O:1
* LIBR2/O:0
?LINK-W-/O or /V option error, re-enter line
*
```

In the above example, the overlay line immediately preceding the error message is ignored, and should be re-entered with an overlay region number greater than or equal to one.

11.6.15 Library List Size Option (/P:n)

The /P:n option lets you change the amount of space allocated for the library routine list. Normally, the default value allows enough space for your needs. It reserves space for approximately 170 unique library routines, which is the equivalent of specifying /P:170. (decimal) or /P:252 (octal). See the *RT–11 Installation Guide* for details on customizing this default number for the library routine list.

The error message ?LINK-F-Library list overflow, increase size with /P indicates that you need to allocate more space for the library routine list. You must relink the program that makes use of the library routines. Use the /P:n option and supply a value for n that is greater than 170 (decimal).

You can use the /P:n option to correct for symbol table overflow. Specify a value for n that is less than 170. This reduces the space used by the library routine list and increases the space allocated for the symbol table. If the value you choose is too small, the ?LINK-F-Library list overflow, increase size with /P message prints.

In the following command, the amount of space for the library routine list is increased to 300 (decimal).

```
* SCCA=RK1:SCCA/P:300.
```

11.6.16 Absolute Base Address Option (/Q)

The /Q option lets you specify the absolute base addresses of up to eight p-sects in your program. This option is particularly handy if you are preparing your program sections in absolute loading format for placement in ROM storage.

When you use this option in the first command line, the linker prompts you for the p-sect names and load addresses. The p-sect name must be six characters or less, and the load address must be an even octal number. Terminate each line with a carriage return. If you enter only a carriage return in response to any of the prompts, LINK ceases prompting.

If you use /E, /Y, or /U with /Q, LINK processes those options before it processes /Q.

When you use the /Q option, observe the following restrictions:

- Enter only even addresses. If you enter an odd address, no address, or invalid characters, LINK prints an error message and then prompts you again for the p-sect and load address.
- /Q is invalid with /H or /R. These options are mutually exclusive.
- LINK moves your p-sects up to the specified address; moving down might destroy code. If your address requires code to be moved down, LINK prints an error message, ignores the p-sect for which you have specified a load address, and continues.

The following example specifies the load addresses for three p-sects.

```
FILE,FT=FILE.FILE1/Q/L (RET)
Load Section:Address?   PSECT1:1000 (RET)
Load Section:Address?   PSECT3:4000 (RET)
Load Section:Address?   PSECT2:2500 (RET)
Load Section:Address?  (RET)
```

11.6.17 REL Format Option (/R[:n])

The /R[:n] option produces an output file in REL format for use as a foreground job with the FB or XM monitor. You cannot use .REL files under the SJ monitor. The /R option assigns the default file type .REL to the output file. The optional argument n represents the amount of stack space to allocate for the foreground job; it must be an even, octal number. The default value is 128 (decimal) bytes of stack space. If you also use the /M option, the value or global symbol associated with it overrides the /R value.

The following command links files FILEI.OBJ and NEXT.OBJ and stores the output on DY1: as FILEO.REL. It also prints a load map on the line printer.

```
* DY1:FILEO,LP:=FILEI,NEXT/R:200
```

You cannot use the /B, /H, or /L option with /R since a foreground REL job has a temporary bottom address of 1000 and is always relocated by FRUN. An error message prints if you attempt this. The /K option is also invalid with /R.

11.6.18 Symbol Table Option (/S)

The /S option instructs the linker to allow the largest possible memory area for its symbol table at the expense of input and output buffer space. Because this makes the linking process slower, you should use the /S option only if an attempt to link a program failed because of symbol table overflow. When you use /S, do not specify a symbol table file or a map in the command string.

11.6.19 Transfer Address Option (/T[:n])

The transfer address is the address at which a program starts when you initiate execution with an R, RUN, SRUN (GET, START), or FRUN command. It prints on the last line of the load map. The /T option lets you specify the start address of the load module. The argument n is a six-digit unsigned, even octal number that defines the transfer address.

If you do not specify n the following message prints:

```
Transfer symbol?
```

In this case, specify the global symbol whose value is the transfer address of the load module. Terminate your response with a carriage return. You cannot specify a number in answer to this message. If you specify a nonexistent symbol, an error message prints and the transfer address is set to 1 so that the program traps immediately if you attempt to execute it. If the transfer address you specify is odd, the program does not start after loading and control returns to the monitor.

Direct assignment (.ASECT) of the transfer address within the program takes precedence over assignment with the /T option. The transfer address assigned with a /T option has precedence over that assigned with an .END assembly directive. To assign the transfer address within a MACRO program, use statements similar to these:

```
          .ASECT
          .=40
          .WORD     START1  ;SYMBOL VALUE FOR TRANSFER ADDRESS
          .PSECT            ;RETURN TO PREVIOUS SECTION

START1:   .
          .
          .
```

or

```
START2:   .                 ;SECONDARY STARTING ADDRESS
          .
          .
          .END      START2
```

The following example links the files LIBR0.OBJ and ODT.OBJ together and starts execution at ODT's transfer address.

```
* LBRODT,LBRODT=LIBR0,ODT/T/W//
* LIBR1/O:1
* LIBR2/O:1
* LIBR3/O:1
* LIBR4/O:1
* LIBR5/O:1
* LIBR6/O:1
* LBREM/O:1//
Transfer symbol?   O.ODT
*
```

11.6.20 Round Up Option (/U:n)

The /U:n option rounds up the section you name in the root so that the size of the root segment is a whole number multiple of the value you specify. The argument n must be a power of 2. When you specify the /U:n option, the linker prompts:

```
Round section?
```

Reply with the name of the program section to be rounded, followed by a carriage return. The program section must be in the root segment. Note that you can round only one program section.

The following example rounds up section CHAR.

```
* LK007,TT:=LK007/U:200
     Round section?  CHAR
```

If the program section you specify cannot be found, the linker prints ?LINK-W-Round section not found AAAAAA and the linking process continues with no rounding.

11.6.21 Extended Memory Overlay Option (/V:n[:m])

Use the /V option to create an extended memory overlay structure for your program. The variable n represents the overlay region number, and m represents a partition number. See Section 11.5.2 for a complete description of this option.

If you use /V on the first command line with no arguments, you enable special .SETTOP features provided by the XM monitor and special .LIMIT features. When used on the first line of the command string, this option allows virtual or privileged foreground or background jobs to map a work area in extended memory with the .SETTOP programmed request. Thus, your program does not need an extended memory overlay structure to make use of the XM .SETTOP features. See the *RT–11 Programmer's Reference Manual* and the *RT–11 Software Support Manual* for more details on these features and extended memory.

11.6.22 Map Width Option (/W)

The /W option directs the linker to produce a wide load map listing. If you do not specify the /W option, the listing is wide enough for three global value columns (normal for paper with 80-character columns). If you use the /W command, the listing is six columns wide, which is suitable for a 132-column page.

11.6.23 Bitmap Inhibit Option (/X)

The /X option instructs the linker not to output the bitmap if code lies in locations 360 to 377 inclusive. This option is provided for compatibility with the RSTS operating system. The bitmap is stored in locations 360–377 in block 0 of the load module, and the linker normally stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory. This information is required by the R, RUN, and GET commands when loading the program; therefore, use care when you use this option.

11.6.24 Boundary Option (/Y:n)

The /Y:n option starts a specific program section in the root on a particular address boundary. Do not use this option with /H. The linker generates a whole number multiple of n, the value you specify, for the starting address of the program section. The argument n must be a power of 2. The linker extends the size of the previous program section to accommodate the new starting address.

When you have entered all the input lines, the linker prompts:

```
Boundary section?
```

Respond with the name of the program section whose starting address you are modifying. Terminate your response with a carriage return. Note that you can specify only one program section for this option. If the program section you specify cannot be found, the linker prints ?LINK-W-Boundary section not found, and the linking process continues.

The RT–11 monitors have internal two-block overlays. The first overlay segment, OVLY0, must start on a disk block boundary:

```
* RT11SJ.SYS=BTSJ,RMSJ,KMSJ,TBSJ/Y:1000
    Boundary Section?  OVLY0
```

11.6.25 Zero Option (/Z:n)

The /Z:n option fills unused locations in the load module and places a specific value in these locations. The argument n represents that value. This option can be useful in eliminating random results that occur when the program references uninitialized memory by mistake. The system automatically zeroes unused locations. Use the /Z:n option only when you want to store a value other than zero in unused locations. You cannot use the R, RUN, FRUN, or GET commands to load into memory a load image block of fill characters.

11.7 Linker Prompts

Some of the linker operations prompt for more information, such as the names of specific global symbols or sections. The linker issues the prompt after you have entered all the input specifications, but before the actual linking begins. Table 11–10 shows the sequence in which the prompts occur.

Table 11–10: Linker Prompting Sequence

Prompt	Option
Transfer symbol?	/T
Stack symbol?	/M
Extend section?	/E:n
Boundary section?	/Y:n
Round section?	/U:n
Load section:address?	/Q
Library search?	/I
Duplicate symbol?	/D

The library search, load section, and duplicate symbol prompts can accept more than one symbol and are terminated by a carriage return in response to the prompt.

Note that if the command lines are in an indirect file and the linker encounters an end-of-file before the prompting information has been supplied, the linker prints the prompt messages on the terminal.

The following example shows how the linker prompts for information when you combine options.

```
* LK001=LK001/T/M/E:100/Y:400/U:20/I/Q/D (RET)
Transfer symbol? 0.ODT (RET)
Stack symbol? ST3 (RET)
Extend section? CHAR (RET)
Boundary section? CODE (RET)
Round section? STKSP (RET)
Load section:address? MAIN:100000 (RET)
Load section:address? (RET)
Library search? $SHORT (RET)
Library search? (RET)
Duplicate symbol? RTN (RET)
Duplicate symbol? (RET)
*
```

Chapter 12
MACRO–11 Assembler Program (MACRO)

This chapter describes how to assemble MACRO–11 programs under the RT–11 operating system.

Output from the MACRO–11 assembler includes any or all of the following:

1. A binary object file – the machine-readable logical equivalent of the MACRO–11 assembly language source code
2. A listing of the source input file
3. A cross-reference file listing
4. A table of contents listing
5. A symbol table listing

To use the MACRO–11 assembler, you should understand how to:

1. Initiate and terminate the MACRO–11 assembler (including how to format command strings to specify files MACRO–11 uses during assembly)
2. Assign temporary work files to nondefault devices, if necessary
3. Use file specification options to override file control directives in the source program
4. Interpret error codes

The following sections describe these topics.

12.1 Calling the MACRO–11 Assembler

To call the MACRO–11 assembler from the system device, respond to the system prompt (a dot printed by the keyboard monitor) by typing:

```
. R MACRO (RET)
```

When the assembler responds with an asterisk (*), it is ready to accept command string input. (You can also call the assembler using the keyboard monitor MACRO command; see Chapter 4 of the *RT–11 System User's Guide* for a description of this command.)

12.2 MACRO–11 Assembler Command String Syntax

The assembler expects a command string consisting of the following items, in sequence:

1. Output file specifications
2. An equal sign (=)
3. Input file specifications

Format this command string as follows (punctuation is required where shown):

dev:obj,dev:list,dev:cref/s:arg = dev:sourcei,...,dev:sourcen/s:arg

where:

dev	is any valid RT–11 device for output; any file-structured device for input.
obj	is the file specification of the binary object file that the assembly process produces; the dev for this file should not be TT or LP.
list	is the file specification of the assembly and symbol listing that the assembly process produces.
cref	is the file specification of the CREF temporary cross-reference file that the assembly process produces. (Omission of dev:cref does not preclude a cross-reference listing, however.)
/s:arg	is a set of file specification options and arguments. (Section 12.5 describes these options and associated arguments.)
sourcei and sourcen	is a file specification for MACRO–11 source files or MACRO library files. (These files contain the MACRO language programs to be assembled. You can specify as many as six source files.)

The following command string calls for an assembly that uses one source file plus the system MACRO library to produce an object file BINF.OBJ and a listing. The listing goes directly to the line printer.

```
* DK:BINF.OBJ,LP:=DK:SRC.MAC
```

All output file specifications are optional. The system does not produce an output file unless the command string contains a specification for that file.

The system determines the file type of an output file specification by its position in the command string. Use commas in place of files you wish to omit. For example, to omit the object file, you must begin the command string with a comma. The following command produces a listing, including cross-reference tables, but not binary object files.

```
* ,LP:/C=(source file specification)
```

You need not include a comma after the final output file specification in the command string.

Table 12–1 lists the default values for each file specification.

Table 12–1: Default File Specification Values

File	Default Device	Default File Name	Default File Type
Object	DK:	Must specify	.OBJ
Listing	Same as for object file	Must specify	.LST
Cref	DK:	Must specify	.TMP
First source	DK:	Must specify	.MAC
Additional source	Same as for preceding source file	Must specify	.MAC
System MACRO library	System device SY:	SYSMAC	.SML
User MACRO library	DK: if first file, otherwise same as for preceding source file	Must specify	.MLB

12.3 Terminating the MACRO–11 Assembler

If you have typed R MACRO and received the asterisk prompt but have not yet entered the command string, you can terminate MACRO–11 control by typing CTRL/C once. After you have completed the command string (thus beginning an assembly) you can halt the assembly process at any time by typing CTRL/C twice. This returns control to the system monitor, and a system monitor dot prompt appears on the terminal.

To restart the assembly process, type R MACRO in response to the system monitor prompt.

12.4 Assigning the Temporary Work File

Some assemblies need more symbol table space than available memory can contain. When this occurs the system automatically creates a temporary work file called WRK.TMP to provide extended symbol table space.

The default device for WRK.TMP is DK. To cause the system to assign a different device, enter the following command:

```
. ASSIGN dev: WF
```

The dev parameter is the physical name of a file-structured device. The system assigns WRK.TMP to this device.

12.5 File Specification Options

At assembly time you may need to override certain MACRO directives appearing in the source programs. You may also need to direct MACRO–11 on the handling of certain files during assembly. You can satisfy these needs by including special options in the MACRO–11 command string in addition to the file specifications. Table 12–2 lists the options and describes the effect of each.

The general format of the MACRO–11 command string is repeated below for your convenience:

dev:obj,dev:list,dev:cref/s:arg = dev:sourcei,...,dev:sourcen/s:arg

Table 12–2: File Specification Options

Option	Function
/C:arg	Control contents of cross-reference listing
/D:arg	Object file function disabling; overrides source program directive .DSABL
/E:arg	Object file function enabling; overrides source program directive .ENABL/L:arg listing control, overrides source program directive .LIST
/L:arg	Listing control; overrides source program directive .LIST
/M	Indicates input file is MACRO library file
/N:arg	Listing control; overrides source program directive .NLIST

The /M option affects only the particular source file specification to which it is directly appended in the command string.

Other options are unaffected by their placement in the command string. The /L option, for example, affects the listing file, regardless of where you place it in the command string.

The following subsections describe how to use the file specification options.

12.5.1 Listing Control Options (/L:arg and /N:arg)

Two options, /L:arg and /N:arg, pertain to listing control. By specifying these options with a set of selected arguments (see Table 12–3) you can control the content and format of assembly listings. You can override at assembly time the arguments of .LIST and .NLIST directives in the source program.

Figure 12–1 shows an assembly listing of a small program. This illustration shows the more important listing features. It labels each feature with the mnemonic ASCII argument that determines its appearance on the listing; the argument SEQ, for instance, controls the appearance of the source line sequence numbers.

Figure 12–1: Sample Assembly Listing

```
.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE 1

SEQ   LOC    BIN     BEX     BEX     BEX     SRC                                  COM
  1           000012                          LF=     012                          ;SYMBOL FOR LINE FEED
  2                                                   .MCALL  .TTYIN, .EXIT
  3                                                   .MACRO  CALL    NAME         ;DEFINE A USER MACRO
  4                                           MD      JSR     PC,NAME
  5                                                   .ENDM
AU  6  000000  000000  000000  000000                 .GLOBAL SUBR1,  SUBR2        ;TWO EXTERNAL SUBROUTINES
  7    000000                                         .CSECT  PROG                 ;DEFINE A CSECT
  8    000000  012702  000050'         START:  MOV     #BUFFER,R2                  ;R2 = ADDS(BUFFER)
AU  9  000004  000000  000000          1$      .TTYIN                              ;READ A CHAR INTO R0
 10    000010  110022                          MOVB    R0,(R2)+                    ;AND STORE IN BUFFER
 11    000012  120027  000012                  CMPB    R0,#LF                      ;WAS IT A LINE FEED?
AU 12  000016  001377                          BNE     1$                          ;NOPE - KEEP READING
 13    000020  105022                          CLRB    (R2)+                       ;ELSE FLAG END OF LINE WITH ZERO
 14    000022  012703  000050'                 MOV     #BUFFER,R3                  ;R3 = ADDS(BUFFER) FOR SUBR1
 15    000026                                  CALL    SUBR1                       ;INVOKE CALL MACRO
U      000026  004767  000000          MC      JSR     PC,SUBR1
 16    000032  103762                          BCS     START                       ;GET A NEW LNE IF CARRY SET
 17    000034                                  CALL    SUBR2                       ;ELSE CALL OTHER SUBR.
U      000034  004767  000000                  JSR     PC,SUBR2
 18    000040  010067  000002                  MOV     R0,ANSWER                   ;AND STORE IN ANSWER
 19    000044                                  .EXIT                               ;RETURN TO RT-11
       000044  104350  <-- MEB                 EXIT    #0350
 20    000046                          ANSWER: .BLKW                               ;DEFINE ANSWER STORAGE
 21    000050                          BUFFER: .BLKB   72.                         ;INPUT LINE BUFFER
 22            000004'                         .END    START

.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE 1-1
SYMBOL TABLE

ANSWER  000046R     002 LF     = 000012          SUBR1 = ******          .GLOBA= ******          .TTYIN= ******
BUFFER  000050R     002 START    000000R     002 SUBR2 = ******
SYM

. ABS.  000000      000
        000006      001
PROG    000160      002
ERRORS DETECTED:  5

VIRTUAL MEMORY USED:  407 WORDS ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR 63 PAGES
,LP:=DT:CRAIG/L:MEB/C:S:F:P:P:M:C  <-- COPY OF COMMAND STRING THAT REQUESTED LISTING
```

Specifying the /N option with no argument causes the system to list only the symbol table, table of contents, and error messages.

Specifying the /L option with no arguments causes the system to ignore .LIST and .NLIST directives that have no arguments.

The following example lists binary code throughout the assembly using the 132-column line printer format, and suppresses the symbol table listing.

```
* I,LP:/L:MEB/N:SYM=FILE
```

Table 12–3: Arguments for /L and /N Listing Control Options

Argument	Default	Listing Control
BEX	List	Binary extensions
BIN	List	Generated binary code
CND	List	Unsatisfied conditionals, .IF and .ENDC statements
COM	List	Comments
LD	No list	List control directives with no arguments
LOC	List	Address location counter
MC	List	Macro calls, repeat range expansion
MD	List	Macro definitions, repeat range expansion
ME	No list	Macro expansions
MEB	No list	Macro expansion binary code
SEQ	List	Source line sequence numbers
SRC	List	Source code
SYM	List	Symbol table
TOC	List	Table of contents
TTM	No list	132-column line printer format when not specified, terminal mode when specified

12.5.2 Function Control Options (/D:arg and /E:arg)

Two options, /E:arg and /D:arg, allow you to enable or disable functions at assembly time, and thus influence the form and content of the binary object file. These functions can override .ENABLE and .DSABL directives in the source program.

Table 12–4 summarizes the acceptable /E and /D function arguments, their normal default status, and the functions they control.

Table 12–4: Arguments for /E and /D Function Control Options

Argument	Default Mode	Function
ABS	Disable	Allows absolute binary output
AMA	Disable	Assembles all absolute addresses as relative addresses
CDR	Disable	Treats all source information beyond column 72 as commentary
CRF	Enable	Allows cross-reference listing; disabling this function inhibits CREF output if option /C is active
FPT	Disable	Truncates floating point values (instead of rounding)
GBL	Enable	Treats undefined symbols as globals
LC	Enable	Allows lowercase ASCII source input
LCM	Disable	Causes the MACRO–11 conditional assembly directives .IF IDN and .IF DIF to sense differences between uppercase and lowercase letters.
LSB	Disable	Allows local symbol block
MCL	Disable	Causes MACRO to search all MACRO libraries for a MACRO definition if an undefined op code is found
PNC	Enable	Allows binary output
REG	Enable	Allows mnemonic definitions of registers

For example, if you type the following commands the system assembles a file while treating columns 73 through 80 of each source line as commentary.

```
. R PIP
* SRCPRG.MAC=CR:/A
* CTRL/C
. R MACRO
* ,LP:=SRCPRG.MAC/E:CDR
```

Because MACRO–11 is a two-pass assembler, you cannot read directly from any non-file-structured device. You must use PIP (or the keyboard monitor COPY command) to transfer input to a file-structured device before beginning the assembly.

Use either the function control or listing control option and arguments at assembly time to override corresponding listing or function control directives in the source program. For example, assume that the sorce program contains the following sequence:

```
. NLIST MEB
  .(MACRO references)
. LIST MEB
```

In this example, you disable the listing of macro expansion binary code for some portion of the code and subsequently resume MEB listing. However, if you indicate /L:MEB in the assembly command string, the system ignores both the .NLIST MEB and the .LIST MEB directives. This enables MEB listing throughout the program.

12.5.3 Macro Library File Designation Option (/M)

The /M option is meaningful only if appended to a source file specification. It designates its associated source file as a macro library.

If the command string does not include the standard system macro library SYSMAC.SML, the system automatically includes it as the first source file in the command string.

When the assembler encounters an .MCALL directive in the source code, it searches macro libraries according to their order of appearance in the command string. When it locates a macro record whose name matches that given in the .MCALL, it assembles the macro as indicated by that definition. Thus if two or more macro libraries contain definitions of the same macro name, the macro library that appears rightmost in the command string takes precedence.

Consider the following command string:

```
*  (output file specification)=ALIB.MLB/M,BLIB.MLB/M,XIZ
```

Assume that each of the two macro libraries, ALIB and BLIB, contain a macro called .BIG, but with different definitions. Then, if source file XIZ contains a macro call .MCALL .BIG, the system includes the definition of .BIG in the program as it appears in the macro library BLIB.

Moreover, if macro library ALIB contains a definition of a macro called .READ, that definition of .READ overrides the standard .READ macro definition in SYSMAC.SML.

12.5.4 Cross-Reference (CREF) Table Generation Option (/C:arg)

A cross-reference (CREF) table lists all or a subset of the symbols in a source program, identifying the statements that define and use the symbols.

12.5.4.1 Obtaining a Cross-Reference Table – To obtain a CREF table you must include the /C:arg option in the command string. Usually you include the /C:arg option with the assembly listing file specification. You can in fact place it anywhere in the command string.

If the command string does not include a CREF file specification, the system automatically generates a temporary file on device DK:. If you need to have a device other than DK: contain the temporary CREF file, you must include the dev:cref field in the command string.

If the listing device is magtape, load the handler for that device before issuing the command string, using the monitor LOAD command (described in Chapter 4 of the *rt–11 System User's Guide*).

A complete CREF listing contains the following six sections:

1. A cross-reference of program symbols; that is, labels used in the program and symbols defined by a direct assignment statement.

2. A cross-reference of register equate symbols. These normally include the symbols R0, R1, R2, R3, R4, R5, SP, and PC, unless the REG function has been disabled through a .DSABL REG directive or the /D:REG option. Also included are any other symbols that are defined in the program by the construct:

   symbol = %n

   where 0^≤n^≤7 and n represents the register number.

3. A cross-reference of MACRO symbols; that is, those symbols defined by .MACRO and .MCALL directives.

4. A cross-reference of permanent symbols, that is, all operation mnemonics and assembler directives.

5. A cross-reference of program sections. These symbols include the names you specify as operands of .CSECT or .PSECT directives. Also included are the default program sections produced by the assembler, the blank p-sect, and the absolute p-sect, .ABS.

6. A cross-reference of errors. The system groups and lists all flagged errors from the assembly by error type.

You can include any or all of these six sections in the cross-reference listing by specifying the appropriate arguments with the /C option. These arguments are listed and described in Table 12–5.

Table 12–5: /C Option Arguments

Argument	CREF Section
C	Control and program sections
E	Error code grouping
M	MACRO symbolic names
P	Permanent symbols including instructions and directives
R	Register symbols
S	User-defined symbols

NOTE

Specifying /C with no argument is equivalent to specifying /C:S:M:E. That special case excepted, you must explicitly request each CREF section by including its arguments. No cross-reference file occurs if the /C option is not specified, even if the command string includes a CREF file specification.

12.5.4.2 Handling Cross-Reference Table Files — When you request a cross-reference listing by means of the /C option, you cause the system to generate a temporary file, DK:CREF.TMP.

If device DK: is write-locked or if it contains insufficient free space for the temporary file, you can allocate another device for the file. To allocate another device, specify a third output file in the command string; that is, include a dev:cref specification. (You must still include the /C option to control the form and content of the listing. The dev:cref specification is ignored if the /C option is not also present in the command string.)

The system then uses the dev:cref file instead of DK:CREF.TMP and deletes it automatically after producing the CREF listing.

The following command string causes the system to use RK2:TEMP.TMP as the temporary CREF file.

```
* ,LP:,RK2:TEMP.TMP=SOURCE/C
```

Another way to assign an alternate device for the CREF.TMP file is to enter the following command prior to entering R MACRO:

```
.ASSIGN dev: CF
```

This method is preferred if you intend to do several assemblies, because it relieves you from having to include the dev:cref specification in each command string. If you enter the ASSIGN dev: CF command, and later include a CREF specification in a command string, the specification in the command string prevails for that assembly only.

The system lists requested cross-reference tables following the MACRO assembly listing. Each table begins on a new page. (Figure 12–2 combines the tables to save space, however.)

The system prints symbols and also symbol values, control sections, and error codes, if applicable, beginning at the left margin of the page. References to each symbol are listed on the same line, left-to-right across the page. The system lists references in the form p-l, where p is the page in which the symbol, control section, or error code appears, and l is the line number on the page.

A number sign (#) next to a reference indicates a symbol definition. An asterisk (*) next to a reference indicates a destructive reference — that is, an operation that alters the contents of the addressed location.

Figure 12–2: Cross-Reference Table

```
.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE S-1
CROSS REFERENCE TABLE (CREF V01-05 )

.GLOBA   1-6
.TTYIN   1-9
ANSWER   1-18#    1-20#
BUFFER   1-8      1-14      1-21#
LF       1-1#     1-11
START    1-9#     1-16      1-22
SUBR1    1-6      1-15
SUBR2    1-5      1-17


.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE R-1
CROSS REFERENCE TABLE (CREF V01-05 )

PC       1-15#    1-17#
R0       1-10     1-11      1-18
R2       1-9#     1-10#     1-13#
R3       1-14#


.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE M-1
CROSS REFERENCE TABLE (CREF V01-05 )

.EXIT    1-2#     1-19
.TTYIN   1-2#
CALL     1-3#     1-15      1-17


.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE P-1
CROSS REFERENCE TABLE (CREF V01-05)

.BLKB    1-21
.BLKW    1-20
.CSECT   1-7
.END     1-22
.MACRO   1-3
.MCALL   1-2
BCS      1-16
BNE      1-12
CLRB     1-13
CMPB     1-11
EMT      1-19
JSR      1-15      1-17
MOV      1-8       1-14      1-18
MOVB     1-10


.MAIN.  MACRO V03.00 6-JUN-77 00:03:57 PAGE C-1
CROSS REFERENCE TABLE (CREF V01-05 )

         0-0
. ABS.   0-0
PROG     1-7


.MAIN.  MACRO V04.00 6-JUN-79 00:03:57 PAGE E-1
CROSS REFERENCE TABLE (CREF V01-05 )

A        1-6       1-9       1-12
U        1-6       1-9       1-12      1-15      1-17
```

12.6 MACRO–11 Error Codes

The MACRO–11 system prints diagnostic error codes as the first character of a source line on which the assembler detects an error. This error code identifies the type of error; for example, a code of M indicates a multiple definition of a label. Table 12–6 shows the error codes that might appear on an assembly listing. For detailed information on error code interpretation and debugging, see the *PDP–11 MACRO–11 Language Reference Manual.*

Table 12–6: MACRO–11 Error Codes

Error Code	Meaning
A	Addressing or relocation error. This code can be generated by any of the following: 1. A conditional branch instruction target that is too far above or below the current statement. Conditional branch targets must be within –128 to –127 (decimal) words of the instruction. 2. A statement that makes an invalid change to the current location counter. For example, a statement that forces the current location counter to cross a .PSECT boundary can generate this code. 3. A statement that contains an invalid address expression. For example, an absolute address expression that has a global symbol, relocatable value, or complex relocatable value can generate this code. The directives .BLKB, .BLKW, and .REPT must have an absolute value or an expression that reduces to an absolute value. 4. Separate expressions in the statement that are not separated by commas. 5. A global definition error. If .ENABL GBL is set, MACRO–11 scans the symbol table at the end of the first pass and marks any undefined symbols as global references. If one of these symbols is subsequently defined in the second pass, a general addressing error occurs. 6. A global assignment statement that contains a forward reference to another symbol. 7. An expression that defines the value of the current location counter and contains a forward reference. 8. An invalid argument for an assembler directive 9. An unmatched delimiter or invalid argument construction.
B	Instruction or word data is being assembled at an odd address. The system increments the location counter by 1, and continues.
D	A nonlocal label is defined more than once, specifically in an earlier statement.
E	The .END assembler directive at the end of the source input is missing. The system supplies an .END statement and completes the current assembly pass.

(Continued on next page)

Table 12–6: MACRO–11 Error Codes (Cont.)

Error Code	Meaning
I	MACRO–11 has detected one or more invalid characters. A question mark (?) replaces each invalid character on the assembly listing, and MACRO–11 continues after ignoring the character.
L	An input line is longer than 132 characters. In particular, this error occurs when the expansion of a macro causes excessive substitution of real arguments for dummy arguments.
M	A label is the same as an earlier label (multiple definition of a label). For example, two labels whose first six characters are identical can generate this error.
N	A number is not in the current program radix. MACRO–11 processes this number as a decimal value.
O	Op-code error. Exceeding the permitted nesting level for conditional assemblies causes this error. Attempting to expand a macro that remains unidentified after an .MCALL search can also generate this code.
P	Phase error. The definition or value of a label differs from one assembler pass to the next, or a local symbol occurs more than once in a local symbol block.
Q	Questionable syntax. For example, missing arguments, too many arguments, or an incomplete instruction scan can generate thi s error code.
R	Register-type error. For example, if the source program attempts an invalid reference to a register, the assembler can gene rate this error code.
T	Truncation error. A number that generates more than 16 bits in a word, or an expression in a .BYTE directive or trap instruction, can cause this error code.
U	Undefined symbol. The assembler assigns the undefined symbol a constant zero value.
Z	Incompatible instruction. This code is a warning that the instruction is not defined for all PDP–11 hardware configurations.

Chapter 13
Peripheral Interchange Program (PIP)

The peripheral interchange program (PIP) is a file transfer and file maintenance utility program. You can use PIP to transfer files between any of the RT–11 devices (listed in Table 3–1 of the *RT–11 System User's Guide*) and to merge, rename, delete, and change the protection status of files.

13.1 Calling and Terminating PIP

To call PIP from the system device, respond to the keyboard monitor prompt (.) by typing:

```
. R PIP (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to type a command string. If you enter only a carriage return at this point, PIP prints its current version number and prompts you again for a command string. You can type CTRL/C to halt PIP and return control to the monitor when PIP is waiting for input from the console terminal. You must type two CTRL/Cs to abort PIP at any other time. To restart PIP, type R PIP or REENTER followed by a carriage return in response to the monitor's dot.

13.2 PIP Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line PIP accepts. You can type as many as six input file names, but only one output file name is allowed. Some of the PIP options accept a date as an argument. The syntax for specifying the date is:

[:dd.][:mmm][:yy.]

where:

dd.	represents the day (a decimal integer in the range 1–31)
mmm	represents the first three characters of the name of the month
yy.	represents the year (a decimal integer in the range 73–99)

The default value for the date is the current system date. If you omit any of the date values (dd, mmm, or yy), the system uses the values from the current system date. For example, if you specify only the year ::82. and the current system date is May 4, 1983, the system uses the date 4.:MAY:82.. If the current date is not set, it is considered 0 (the same as for an undated file in a directory listing).

On random-access devices such as disks, and in transfers from magtape, PIP operations retain a file's creation date. If the file's creation date is 0, PIP gives it the current system date. However, in transfers to magtape, PIP always gives files the current system date.

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, the system prints -BAD- in the creation date column of each file created beyond the end-of-month. (Note that you can eliminate -BAD- by using the RENAME/SETDATE command after you set the date.)

If you specify a command involving random-access devices for which the output specification is the same as the input specification, PIP does not move any files. However, it can change the creation dates on the files if you use /T, it can rename the files if you use /R, it can protect files if you use /F, or it can remove protection from files if you use /Z.

Because PIP performs file transfers for all RT–11 data formats (ASCII, object, and image), it does not assume file types for either input or output files. You must explicitly specify all file types where file types are applicable.

13.3 Using Wildcards with PIP

You can use all variations of the wildcard construction for the input file specifications in the PIP command line (Section 4.2 of the *RT–11 System User's Guide* describes wildcard usage). Output file specifications cannot contain embedded wildcards. If you use any wild character in an input file specification, the corresponding output file name or file type must be an asterisk. (The concatenate copy operation is an exception to this rule because it does not allow wildcards in the output specification.) The following example shows wildcard usage:

```
***.B=A%B.MAC
```

In this example, the embedded percent character (%) represents any single, valid file name character. In the output file specification, the asterisk represents any valid file name.

The following command deletes all files with the file type .BAK (regardless of their file names) from device DK:.

```
***.BAK/D
```

The next command renames all files with a .BAK file type (regardless of file names) so that these files now have a .TST file type (maintaining the same file names).

```
* *.TST=*.BAK/R
```

In most cases, PIP performs operations on files in the order in which they appear in the device directory. In transfers from magtape (and for all other transfers requested on the same command line), PIP performs operations on files in the order in which they appear on the volume. When you use wildcards in the input file types, PIP ignores system files with the file type .SYS unless you also use the /Y option. PIP prints the error message ?PIP-W-No .SYS action if you omit the /Y option on a command that would operate on .SYS files.

NOTE

> You cannot perform any operations that result in deleting a protected file. For example, you cannot transfer a file to a volume if a protected file with the same name already exists on the output volume.

PIP ignores all files with the file type .BAD unless you explicitly specify both the file name and file type in the command string. PIP does not print a warning message when it does not include .BAD files in an operation.

This example transfers all files, including system files (regardless of file name or file type) from device DK: to device DL1:. It does not transfer .BAD files.

```
* DL1:*.*/Y=*.*
```

13.4 Options

PIP options, summarized in Table 13–1, permit you to perform various operations with PIP. If you do not specify an option, PIP assumes that the operation is a file transfer in image mode. You can put command options at the end of the command string or type them after any file name in the string. Operations involving magtape are an exception, because the /M option is device-dependent and has a different meaning when you specify it on the input or output side of a command line. Type any number of options in a command line, as long as only one operation (copy, delete, or rename) is represented. You can, however, combine copy and delete operations on one line. Also, the protect and noprotect options can be combined with copy and rename operations.

Table 13–1: PIP Options

Option	Section	Function
/A	13.4.2.2	Copies files in ASCII mode, ignoring and discarding nulls and rubouts. It converts input file to 7-bit ASCII and treats CTRL/Z (32 octal) as the logical end-of-file on input (the default copy mode is image).
/B	13.4.2.3	Copies files in formatted binary mode (the default copy mode is image).
/C[:date]	13.4.3	Used with other options to include only files with the specified date in the operation. If you use /C and do not specify a date, PIP includes only files with the current date in the specified operation.
/D	13.4.4	Deletes input files from a specific device. Note that PIP does not automatically query before it performs the operation. If you combine /D with a copy operation, PIP performs the delete operation after the copy completes. This option is invalid in an input specification with magtape.
/E	13.4.5	Transfers files in a single- or small-disk system. PIP initiates the transfer, but pauses and waits for you to mount the volumes involved in the transfer.
/F	13.4.6	Protects files from deletion. Gives protected status to output files during a copy operation so you cannot delete them. If you use neither /F nor /Z, the output files retain the protection status of the input files. Can also be used with /R. Invalid for magtapes.
/G	13.4.7	Ignores any input errors that occur during a file transfer and continues copying.
/H	13.4.8	Verifies that the output file matches the input file after a copy operation. Cannot be used with /A or /B.
/I[:date]	13.4.9	Used with other options to include only files created on or after the specified date.
/J[:date]	13.4.10	Used with other options to include only files created before the specified date.
/K:n	13.4.11	Makes n copies of the output files to any sequential device, such as LP:, TT:, or PC:.
/M:n	13.4.1	Used when I/O transfers involve magtape.
/N	13.4.12	Does not copy or rename a file if a file with the same name exists on the output device. This option protects you from accidentally deleting a file. It is invalid for magtape in the output specification.
/O	13.4.13	Deletes a file on the output device if you copy a file with the same name to that device. The delete operation occurs before the copy operation. This option is invalid for magtape in the output specification.
/P	13.4.14	Copies or deletes all files except those you specify.

(Continued on next page)

Table 13–1: PIP Options (Cont.)

Option	Section	Function
/Q	13.4.15	Use only with another operation. The /Q option causes PIP to print the name of each file to be included in the operation you specify. You must respond with a Y to include a particular file.
/R	13.4.16	Renames the file you specify. This operation is invalid for magtape.
/S	13.4.17	Copies files one block at a time.
/T[:date]	13.4.18	Puts the specified date on all files involved in the operation. This option is invalid when copying to magtape; operations involving magtape devices always use the current date.
/U	13.4.19	Copies and concatenates all files you specify.
/V	13.4.20	Copies files from one input volume to two or more smaller output volumes.
/W	13.4.21	Prints on the terminal a log of all files involved in the operation.
/X	13.4.22	Causes PIP to print an information message instead of a fatal message when it cannot find a file you specified in the command line.
/Y	13.4.23	Includes .SYS files in the operation you specify. You cannot modify or delete these files unless you use the /Y option when you use wildcards in the input file types.
/Z	13.4.24	Removes protected status from output files so you can delete them. If you use neither /F nor /Z, the output files retain the protection status of the input files. When used with /R, enables files for deletion if they have been previously protected with /F. Invalid for magtapes.

13.4.1 Operations Involving Magtape (/M:n)

PIP handles magtape, which is a sequential-access device, differently from random-access devices, such as disks, diskettes, and DECtape II. On magtape, files are stored serially, one after another, and there is no directory at the beginning of each device that lists the files and gives their location. Thus, you can access only one file at a time on each sequential-access device unit. Avoid commands that specify the same device unit number for both the input and output files — they are invalid.

The /M:n option makes operations that involve magtape more efficient. This option lets you specify different tape handling procedures for PIP to follow. The following sections outline the operations that involve magtape and describe the different procedures for using these devices that you can specify with the /M:n option. Remember that when you use the /M:n option, n is interpreted as an octal number. You must use n. (n followed by a decimal point) to represent a decimal number.

Magnetic tape is a convenient auxiliary storage medium for large amounts of data, and is often used as backup for disks. Reflective strips indicate the beginning and end of the tape. A special label (an EOF1 or EOV1 tape label) followed by two tape marks indicates the end of current data and also where new data can begin.

The following PIP options are valid for use with magtape: /A, /B, /C[:date], /F, /G, /H, /I, /J, /M, /P, /Q, /S, /U, /V (only when magtape is the output volume), /W, /X, /Y, and /Z. These options are invalid with magtape: /E, /K, /R, /T, and /V (when magtape is the input volume). The /M:n option lets you direct the tape operation; you can move the tape and perform an operation at the point you specify. Note that /D is invalid for input from magtape; /N and /O are invalid for output to magtape.

The /M:n option can be different for the output and input side of the command line. Since the option applies to the device and not to the files, you can specify one /M:n option for the output file and one for each input file.

Sometimes PIP begins an operation at the current position. To determine the current position, the magtape handler backspaces from its present position on the tape until it finds either an EOF indicator or the beginning of tape (BOT), whichever comes first. PIP then begins the operation with the file that immediately follows the EOF or BOT. The magtape handler also has a special procedure for locating a file with sequence number n:

1. If the file sequence number is greater than the current position, PIP searches the tape in the forward direction.
2. If the file sequence number is more than one file before the current position, or if the file sequence number is less than five files from BOT, the tape rewinds before PIP begins its search.
3. If the file sequence number is at the current position, or if it is one file past the current position, PIP searches the tape in the reverse direction.

Whenever you fetch or load a new copy of the magtape handler, the tape position information is lost. The new handler searches backward until it locates either BOT or a label from which it can learn the position of the tape. It then operates normally, according to steps 1, 2, and 3 described above.

If you omit the /M:n option, the tape rewinds between each operation. Using /M:0 has the same effect as omitting /M:n. When n is positive, it represents the file sequence number. When n is negative, it represents an instruction to the magtape handler.

In copying from magtapes, /M:n functions as follows:

1. If n is 0:

   The tape rewinds and PIP searches for the file you specify. If you specify more than one file, the tape rewinds before each search. If the file specification contains a wildcard, the tape rewinds only once and then PIP copies all the appropriate files.

2. If n is a positive integer:

   PIP goes to file sequence number n. If the file it finds there is the one you specified, PIP copies it. Otherwise, PIP prints the ?PIP-F-File not found DEV:FILNAM.TYP message. If you use a wildcard in the file specification, PIP goes to file sequence number n and then begins to search for matching files.

3. If n is –1:

   PIP starts the search at the current position. If the current position is not the beginning of the tape, PIP may not find the file you specify, even though it does exist on the tape.

In writing to magtapes, /M:n functions as follows:

1. If n is 0:

   The tape rewinds before PIP copies each file. PIP prints a warning message if it finds a file with the same name and file type as the input file and does not perform the copy operation.

2. If n is a positive integer:

   PIP goes to the file sequence number n and enters the file you specify. If PIP reaches logical end-of-tape (LEOT) before it finds file sequence number n, it prints the ?PIP-F-File sequence number not found message. If you specify more than one file or if you use a wildcard in the file specification, the tape does not rewind before PIP writes each file, and PIP does not check for duplicate file names.

3. If n is –1:

   PIP goes to the LEOT and enters the file you specify. It does not rewind, and it does not check for duplicate file names.

4. If n is –2:

   The tape rewinds between each copy operation. PIP enters the file at LEOT or at the first occurrence of a duplicate file name.

If PIP reaches the physical end-of-tape before it completes a copy operation, it cannot continue the file on another tape volume. Instead, it deletes the partial file by backspacing and writing a logical end-of-tape over the file's header label. You must restart the operation and use another magtape.

If you type consecutive CTRL/Cs during any output operation to magtape, PIP does not write a logical end-of-tape at the end of the data. Consequently, you cannot transfer any more data to the tape unless you follow one of the following recovery procedures.

1. Transfer all good files from the interrupted tape to another tape and initialize the interrupted tape in the following manner:

```
* dev1:*.*=dev0:*.*
_ CTRL/C
. R DUP
* dev0:/Z/Y
```

2. Determine the sequential number of the file that was interrupted and use the /M:n construction to enter a replacement file (either a new file or a dummy) over the interrupted file. PIP writes the replacement file and a good LEOT after it. The following example assumes the bad file is the fourth file on the tape:

```
* dev0:file.new/M:4=file.dum
```

13.4.2 Copy Operations

PIP copies files in image, ASCII, and binary format. Other options let you change the date on the files, access .SYS files, combine files, change a file's protection status, and perform other similar operations. PIP automatically allocates the correct amount of space for new files in copy operations. For block-replaceable devices, PIP stores the new file in the first empty space large enough to accommodate it. If an error occurs during a copy operation, PIP prints a warning message, stops the copy operation, and prompts you for another command. You cannot copy .BAD files unless you specifically type each file name and file type.

13.4.2.1 Image Mode – If you enter a command line without an option, PIP copies files onto the destination device in image mode. Note that you cannot reliably transfer memory image files to the line printer or console terminal. PIP can image-copy ASCII and binary data but it does not do any of the data checking described in Section 13.4.2.3.

The following command makes a copy of the file named XYZ.SAV on device DK: and assigns it the name ABC.SAV. (Both files exist on device DK: after the operation.)

```
* ABC.SAV=XYZ.SAV
```

The next example copies from DK: all .MAC files whose names are three characters long and begin with A. PIP stores the resulting files on DY1:.

```
* DY1:*.*=A%%.MAC
```

13.4.2.2 ASCII Mode (/A) – Use the /A option to copy files in 7-bit ASCII mode. PIP ignores and eliminates nulls and rubouts during file transfer. PIP treats CTRL/Z (32 octal) as logical end-of-file if it encounters that character in the input file. You cannot use the /A option with the /V option.

The following command copies F2.FOR from device DK: onto device DY1: in ASCII mode and assigns it the name F1.FOR.

```
* DY1:F1.FOR=F2.FOR/A
```

13.4.2.3 Binary Mode (/B) – Use the /B option to transfer formatted binary files (such as .OBJ files produced by the assembler or the FORTRAN compiler and .LDA files produced by the linker). You cannot use the /B option with the /V option.

The following command transfers a formatted binary file from device DL0: to device DK: and assigns it the name FILE.OBJ.

```
* DK:FILE.OBJ=DL:/B
```

When performing formatted binary transfers, PIP prints a warning if a checksum error occurs. If there is a checksum error and you did not use /G to ignore the error, PIP does not perform the copy operation. You cannot copy library files with the /B option. Copy library files in image mode.

13.4.3 Date Option (/C[:date])

The /C[:date] option includes only those files with the specified date. If no date is specified only those files with the current date are included. Specify /C only once in the command line; it applies to all the file specifications in the entire command.

The following command copies (in ASCII mode) all files with the file type .MAC on DL0: that also have the date January 12, 1983. It also copies the file RDWR.MAC, if it has the date January 12, 1983, from DY0: to DY1:. It combines all these files under the name NN3.MAC on DY1:.

```
* DY1:NN3.MAC=DL0:*.MAC/C:12.:JAN:83.,DY0:RDWR.MAC/A/U
```

The next command copies all files with the current date (except .SYS and .BAD files) from DK: to DY1:. This is an efficient way to back up all new files after a session at the computer.

```
* DY1:*.*=*.*/C
```

13.4.4 Delete Option (/D)

Use the /D option to delete one or more files from the device you specify. Note that PIP does not query you before it performs this operation unless you use /Q. Remember to use the /Y option to delete .SYS files if you use wildcards in the input file types. You cannot delete .BAD files, unless you name each one specifically, including file name and file type. You can specify only six files in a delete operation unless you use wildcards. You must always indicate a file specification in the command line. A delete command consisting only of a device name (dev:/D) is invalid. The delete option is also invalid for magtape.

The following examples illustrate the delete operation.

```
* FILE1.SAV/D
```

The command shown above deletes FILE1.SAV from device DK:.

```
*  DY1:*.*/D
?PIP-W-No .SYS action
*
```

The command shown above deletes all files from device DY1: except those with a .SYS or .BAD file type. Since there is a file with a .SYS file type, PIP prints a warning message to remind you that this file has not been deleted.

```
*  *.MAC/D
```

This command deletes all files with a .MAC file type from device DK:.

13.4.5 Wait Option (/E)

If you have a single-disk system or a diskette system, you will find the /E option useful for copy operations. Use this option when you need to change storage volumes during a copy procedure. The general format of the command line follows.

filespec/E = filespec

You can use any option with /E that is valid with your RT–11 configuration. You cannot use wildcards as input. When you use /E, make sure that PIP is on your system volume.

When you use the /E option, PIP guides you through a series of steps in the process of completing the file transfer. PIP initiates execution of the command, but then pauses and prints the message Mount input volume in <device>; Continue?, where <device> represents the device into which you mount the input volume. At this time you can remove the system volume (if necessary) and mount the volume on which you actually want the operation to take place.

When the new volume is loaded, type Y or any string beginning with Y followed by a carriage return to execute the operation. If you type N or any string beginning with N, or CTRL/C, the operation is not completed. Instead PIP prompts you to remount the system volume if you have removed it and the monitor prompt (.) appears. Any other response causes the message to repeat.

If you type Y, PIP prompts you for the input volume, if any. When the operation completes the Mount system volume in <device>; Continue? message prints. Replace the system device and type Y or any string beginning with Y followed by a carriage return. If you type any other response, PIP prompts you to mount the system volume until you type Y. When you type Y, the asterisk (*) prompt prints, and PIP waits for you to enter another command.

The sections that follow describe the procedures for single-drive and double-drive transfer.

13.4.5.1 Single-Drive Operation – If you want to transfer a file between two storage volumes, and you have only one drive for that type of storage volume, follow the procedure below.

1. Enter a command string according to this general syntax:

   *output-filespec/E = input-filespec

   where output-filespec represents the destination device and file specification, and input-filespec represents the source device and file specification.

2. PIP responds by printing the following message at the terminal.

   ```
   Mount input volume in <device>; Continue?
   ```

   where <device> represents the device into which you are to mount your input volume. Type Y or any string beginning with Y followed by a carriage return after you have mounted your input volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

3. PIP continues the copy procedure and prints the following message on the terminal:

   ```
   Mount output volume in <device>; Continue?
   ```

   After you have removed your input volume from the device, mount your output volume and type Y or any string beginning with Y followed by a carriage return. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

4. Depending on the size of the file, PIP may repeat the transfer cycle (steps 2 and 3) several times before the transfer is complete. When the transfer is complete, PIP prints the following message if you had to remove the system volume from <device>:

   ```
   Mount system volume in <device>; Continue?
   ```

   When you remount the system volume and type Y or any string beginning with Y followed by a carriage return in response to the last instruction, you complete the copy operation. If you type anything other than Y, PIP continues to prompt you to remount the system volume until you type Y.

13.4.5.2 Double-Drive Operation – You can use the /E option for transferring files between two nonsystem volumes. The procedure for transferring files this way follows.

1. With your system volume mounted, enter a command string according to the following general syntax:

   output-filespec/E = input-filespec

   where output-filespec represents the destination device and file specification, and input-filespec represents the source device and file specification.

2. After you have entered the command string, PIP responds with the message:

   ```
   Mount input volume in <device>; Continue?
   ```

   Type Y or any string beginning with Y followed by a carriage return when you have mounted the input volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

3. PIP then prints:

   ```
   Mount output volume in <device>; Continue?
   ```

   Type Y or any string beginning with Y followed by a carriage return after you have mounted the output volume. If you type any string beginning with N or if you type CTRL/C, the operation is not performed and the monitor prompt (.) appears. If you have removed the system volume, PIP prompts you to remount it.

4. Unlike the single-volume transfer, the double-volume transfer involves only one cycle of mounting the input and output volumes. When the file transfer is complete, PIP prints the following message if you had to remove the system volume from <device>:

   ```
   Mount system volume in <device>; Continue?
   ```

   When you type Y or any string beginning with Y followed by a carriage return in response to the last instruction, you complete the copy operation. If you type anything other than Y, PIP continues to prompt you to mount the system volume until you type Y.

13.4.6 Protection Option (/F)

Use the /F option to protect files. The letter P next to the block size number in the file's directory entry indicates the file is protected.

If a file is protected you cannot perform any operations on it that result in deleting the file. You can copy a protected file to another volume or change its name. However, you cannot change its protected status unless you use the /Z (no protection) option. Note that the contents of a protected file are not protected; that is, although you cannot delete a protected file, you can change or delete its contents.

You can also use the /F option during copy operations to protect the output file, and with /R to change a file's protection status. If during a copy operation you use neither /F nor /Z, the output files retain the protection status of the input files.

The following command protects all files with the file type .MAC on DK:.

```
**.MAC/F
```

The following command copies all files with file type .ORI from DL0: to DL1:. The resulting output files on DL1: are protected from deletion.

```
*DL1:*.*=DL0:*.ORI/F
```

If you use the /F option with a file that is already protected, no operation is performed on that file regardless of any other options in the command string. For example, the following command requests PIP to protect the file DY1:CALCAB.MAC and change its creation date to April 21, 1983. However, because the file is already protected, PIP performs neither operation.

```
*DL1:CALCAB.MAC/F/T:21.:APR:83.
```

13.4.7 Ignore Errors Option (/G)

The /G option copies files, but ignores all input errors. This option forces a single-block transfer, which you can invoke at any other time with the /S option. Use the /G option if an input error occurred when you tried to perform a normal copy operation. The procedure can sometimes recover a file that is otherwise unreadable. If an error still occurs, PIP prints the ?PIP-W-Input error DEV:FILNAM.TYP message and continues the copy operation.

The following command, copies the file TOP.SAV in image mode from device DY1: to device DK: and assigns it the name ABC.SAV.

```
*ABC.SAV=DY1:TOP.SAV/G
```

The next command copies files F1.MAC and F2.MAC in ASCII mode from device DY0: to device DY1:. This command creates one file with the name COMB.MAC, and ignores any errors that occur during the operation.

```
*DY1:COMB.MAC=DY0:F1.MAC,F2.MAC/A/G/U
```

13.4.8 Verify Option (/H)

Use the /H option to verify that the output file matches the input file when a copy operation is performed. If the two files are different a message is printed on the terminal. This option cannot be used with /A or /B.

The following command verifies that the output file A.BAK on DY1: is the same as the input file A.MAC on DY0:.

```
*DY1:A.BAK=DY0:A.MAC/H
```

13.4.9 Since Option (/I[:date])

The /I[:date] option includes only those files created on or after the specified date. If no date is specified, PIP uses the current date.

The following command copies from DK: only those .MAC files created on or after January 4, 1983:

```
*DL0:*.MAC=*.MAC/I:4.:JAN:83.
```

13.4.10 Before Option (/J[:date])

The /J[:date] option includes only those files created before the specified date. If you do not specify a date, PIP uses the current date.

The following command copies only those .MAC files created before January 14, 1983:

```
*DL0:*.MAC=*.MAC/J:14.:JAN:83.
```

13.4.11 Copies Option (/K:n)

The /K:n option directs PIP to generate n copies of the file you specify. The only valid output devices are the console terminal and the line printer. Normally, each copy of the file begins at the top of a page; copies are separated by form feeds.

```
*LP:=STOTLE.LST/K:3
```

This command, for example, prints three copies of the listing file, STOTLE.LST, on the line printer.

13.4.12 No Replace Option (/N)

The /N option prevents execution of a copy or rename operation if a file with the same name as the output file already exists on the output device. This option is not valid when output is to magtape.

The following example uses the /N option.

```
*DY0:CT.SYS=DK:CT.SYS/Y/N
?PIP-W-Output file found, no operation performed DK:CT.SYS
*
```

The file named CT.SYS already exists on DY0:, and the copy operation does not proceed.

13.4.13 Predelete Option (/O)

The /O option deletes a file on the output device if you copy a file with the same name to that device. PIP deletes the file on the output device before the

copy operation occurs. Normally, PIP deletes a file of the same name after the copy completes. This option is not valid when output is to magtape.

The following example uses the /O option.

```
*DL1:TES11.MAC=DY1:TEST.MAC/O
```

If a file named TEST1.MAC already exists on DL1:, PIP deletes it before copying TEST.MAC from DY1: to TEST1.MAC on DL1:.

13.4.14 Exclude Option (/P)

The /P option directs PIP to include all files in the operation except the ones you specify. Note that if you want to include system (.SYS) files and you use the /P option, you must always use the /Y option with it.

```
* DY0:*.*=DY1:*.MAC/P
```

This command directs PIP to transfer all files from DY1: to DY0: except the .MAC files. The .SYS files will also be excluded from the operation because the /Y option was not specified.

13.4.15 Query Option (/Q)

Use the /Q option with another PIP operation to list all files and to request confirmation for each file before it is included. Typing Y or any string beginning with Y followed by a carriage return causes the named file to be processed; typing anything else excludes the file.

The following example deletes four files from DY1:.

```
* DY1:*.*/D/Q
 Files deleted:
DY1:FIX463.SAV?
DY1:GRAPH.BAK ?  Y
DY1:DMPX.MAC  ?
DY1:MATCH.BAS ?
DY1:EXAMP.FOR ?
DY1:GRAPH.FOR ?  Y
DY1:GLOBAL.MAC?  Y
DY1:PROSEC.MAC?  Y
DY1:KB.MAC    ?
DY1:EXAMP.MAC ?
*
```

13.4.16 Rename Operation (/R)

Use the /R option to rename a file you specify as input, giving it the name you specify in the output specification. The input and output volumes for a rename operation must be the same. PIP prints an error message if the command specifications are not valid. Use the /Y option if you rename .SYS files and you use wildcards in the input file types. You cannot use /R with magtape.

The following examples illustrate the rename operation.

```
* DY1:F1.MAC=DY1:F0.MAC/R
```

The command shown above renames F0.MAC to F1.MAC on device DY1:.

```
* DL1:OUT.SYS=DL1:CT.SYS/R
```

This command renames file CT.SYS to OUT.SYS.

The rename command is particularly useful when a file contains bad blocks. By giving the file a .BAD file type, you can ensure that the file permanently resides in that area of the device. Thus, the system makes no other attempts to use the bad area. Once you give a file a .BAD file type, you cannot move it during a compress operation. You cannot rename .BAD files unless you specifically indicate both the file name and file type.

13.4.17 Single-Block Transfer Option (/S)

The /S option directs PIP to copy files one block at a time. On some devices, this operation increases the chances of an error-free transfer. You can combine the /S option with other PIP copy options. For example:

```
* DL1:TEST.MAC=DL0:TEST.MAC/S
```

PIP performs this transfer one block at a time.

13.4.18 Set Date Option (/T[:date])

This option causes PIP to put the specified date on all files involved in the operation. If you specify no date, PIP uses the current system date. Normally, PIP preserves the existing file creation date on copy and rename operations. This option is invalid when copying to magtape, because PIP always uses the current date for these operations.

The following command copies all the files with file type .COM copied from DY0: to DY1:, and assigns the output files the date January 24, 1983.

```
* DY1:*.*=DY0:*.COM/Y/T:24.:JAN:83.
```

13.4.19 Concatenate Option (/U)

To combine more than one file into a single file, use the /U option. This option is particularly useful when you want to combine several object modules into a single file for use by the linker or librarian. PIP does not accept wildcards on the output specification. Use the /B option with /U if you are concatenating object (.OBJ) files.

The following examples show the /U option.

```
* DK:AA.OBJ=DY1:BB.OBJ,CC.OBJ,DD.OBJ/U/B
```

The command shown above transfers files BB.OBJ, CC.OBJ, and DD.OBJ to device DK: as one file and assigns it the name AA.OBJ.

```
*DL1:MERGE.MAC=DL0:FILE2.MAC,FILE3.MAC/A/U
```

This command merges ASCII files FILE2.MAC and FILE3.MAC on DL0: into one ASCII file named MERGE.MAC on device DL1:.

13.4.20 Multivolume Option (/V)

The /V option copies files from an input volume to two or more smaller output volumes. This option is useful when you are copying several files from a large input volume and you are not sure whether all the files will fit on one output volume.

When you use this option PIP copies files to the output volume until the system finds a file that will not fit. PIP continues to search that file's directory segment, copying all files from the segment that will fit onto the output volume. When no more files from that segment will fit on the output volume, PIP prompts you to mount the next output volume and prints the Continue? message. Mount another output volume of the same type and type Y or any string beginning with Y followed by a carriage return to continue the copy operation. If you type any string beginning with N or if you type CTRL/C, the operation is aborted and the monitor prompt (.) appears.

When you type Y to continue, PIP copies the first file that would not fit to the previous output volume to the new output volume. PIP continues to copy files from that directory segment until no more files from that segment will fit on the output volume or until all files from that directory segment have been copied. If all files from that segment have been copied, PIP begins copying files from the next directory segment. File copying continues in this fashion until all the specified input files have been copied.

The following example copies all files on DL0: to several double-density diskettes:

```
*[DY0:*.*=DL0:*.*/V
Mount next output volume in DY0:; Continue? Y
Mount next output volume in DY0:; Continue? Y
Mount next output volume in DY0:; Continue? Y
*
```

13.4.21 Logging Option (/W)

When you use the /W option, PIP prints a list of all files included in the operation. The /W option is useful if you do not want to take the time to use the query mode (the /Q option, described in Section 13.4.15), but you do want a list of the files operated on by PIP.

PIP prints the log for an operation on the terminal under the command line. This example shows logging with the delete operation.

```
*L DY1:*.*/D/W
?PIP-W-No .SYS action
 Files deleted:
DY1:TEST.MAC
DY1:FIX463.SAV
DY1:GRAPH.BAK
DY1:DMPX.MAC
DY1:MATCH.BAS
DY1:EXAMP.FOR
DY1:GRAPH.FOR
DY1:GLOBAL.MAC
DY1:PROSEC.MAC
DY1:EXAMP.MAC
*
```

13.4.22 Information Option (/X)

The /X option causes PIP to print an information message when PIP fails to find all of the files you specify in a command line. If you do not use the /X option, PIP prints a fatal error message when it is unable to find an input file, and control returns to the keyboard monitor after the operation completes. Use /X in indirect command files to ensure that processing will continue even if PIP fails to find a file you specify.

In the following example, the input files FILE1.TXT and FILE3.TXT are copied to DL1:. However, since the system is unable to find DL0:FILE2.TXT, PIP prints a message to inform you.

```
* DL1:*.*=DL0:FILE1.TXT,FILE2.TXT,FILE3.TXT
?PIP-I-File not found DL0:FILE2.TXT
```

13.4.23 System Files Option (/Y)

Use the /Y option if you need to perform an operation on system (.SYS) files and you use wildcards in the input file type. For example:

```
* *.*=DY1:*.*/Y
```

This command copies to device DK:, in image mode, all files (including .SYS files) from device DY1:. Note that you must always use /Y with the /P option to include .SYS files, even when you use no wildcards.

13.4.24 No Protection Option (/Z)

Use the /Z option to remove protected status from files, so that you can delete or change those files. You can also use the /Z option with /R to change the protection status of a file, and during copy operations to remove protection from the output file.

Note that since you cannot delete files assigned as logical disks, you cannot use the /Z option to remove protection from these files.

The following command removes protection from all .MAC files on DK:.

```
* *.MAC/Z
```

The following command copies the file PROGRM.MAC from DY0: to DY1:. The resulting output file on DY1: is enabled for deletion.

```
* DY1:PROGRM.MAC=DY0:PROGRM.MAC/Z
```

If you use the /R option with a file that is already unprotected, no operation is performed on that file regardless of any other options in the command string. For example, the following command requests PIP to unprotect the file DY1:CALCAB.MAC and change its creation date to April 21, 1983. However, because the file is already unprotected, PIP performs neither operation.

```
* DL1:CALCAB.MAC/R/T:21.:APR:83.
```

Chapter 14
Resource Utility Program (RESORC)

The resource utility program lists system resource information on the terminal. You can use RESORC to display the following data about your system:

- Monitor version number
- SET options in effect
- Hardware configuration
- Total amount of memory in system
- Organization of physical memory
- Currently loaded jobs
- System generation special features in effect
- Device assignments
- Status of currently active terminals (on multiterminal systems)
- Device handler status
- Logical disk subsetting information

14.1 Calling and Terminating RESORC

To call RESORC from the system device, respond to the keyboard monitor dot (.) by typing:

```
. R RESORC (RET)
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for your input. At this point, enter the RESORC option or options required to obtain the information you need. Section 14.2 describes these options, and Table 14–1 summarizes them.

If you enter only a carriage return in response to the asterisk, RESORC prints its name and current version number. To abort RESORC while it is executing, type two CTRL/Cs. Type one CTRL/C to return control to the monitor when RESORC is waiting for input (that is, when an asterisk has printed on the terminal).

14.2 Options

By specifying one or more of the options /C, /D, /H, /J, /L, /M, /O, /S, /T, and /X, you choose the information that RESORC lists on the terminal. If you use two or more options, you can enter them in any order, although RESORC lists the information in the order /M, /C, /H, /O, /D, /L, /J, /T, /X, /S.

RESORC offers two additional options that are equivalent to combinations of options. The /Z option has the same effect as a combination of the /M, /C, /H, /J, and /O options. The /A option has the same effect as a combination of all the options except /Z.

Table 14–1: RESORC Options

Option	Display
/A	The result of a combination of all RESORC options (except /Z)
/C	The device from which you bootstrapped the system and the monitor SET options in effect
[dd:]/D	A list of the device handlers, their status, and their vectors; when [dd:] is included, lists the status of only that device
/H	The hardware configuration, including the system's total amount of memory (in bytes)
/J	Information about the currently running and loaded jobs
/L	Device assignments
/M	The monitor type, version number, and update level
/O	The system generation special features in effect
/S	Information about logical disk subsetting
/T	The status and options in effect for currently active terminals on multiterminal systems
/X	The organization of physical memory
/Z	The result of a combination of /M, /C, /H, /J, and /O options

14.2.1 All Option (/A)

The /A option has the same effect as a combination of all the other RESORC options (except /Z). When you enter /A, all RESORC information is printed on the terminal in the order shown below.

1. Monitor configuration (that is, the monitor type and version number; the device from which the monitor was bootstrapped; what the SET options are; whether a foreground job is loaded; and the indirect file nesting depth)
2. Hardware configuration

3. System generation special features in effect
4. Device handler status
5. Device assignments
6. Job status
7. Status of currently active terminals
8. Organization of physical memory
9. Subsetting of physical disks into logical disks

See the following sections for details on these topics.

14.2.2 Software Configuration Option (/C)

The /C option displays:

1. The device from which you bootstrapped the system
2. The monitor SET options
3. Whether a foreground job is loaded (if applicable)
4. Indirect file nesting depth

The following example uses the /C option.

```
* /C

RT-11FB(S)  V05.00
Booted from DL1:

USR is set SWAP
EXIT is set SWAP
KMON is set IND
TT is set QUIET
ERROR is set ERROR
KMON nesting depth is 3
```

14.2.3 Device Handler Status Option ([dd:]/D)

RESORC's /D option displays a list of your system's device handlers, along with their status, CSR addresses, and vectors. The /D option lists only those handlers whose special features match those of the currently booted monitor. If a handler is loaded, RESORC prints its load address.

If you specify a device name in front of the /D option (dd:/D), RESORC lists information for only that device. The variable dd represents the 2-letter permanent device name for the device (see Table 3–1 in the *RT–11 System User's Guide*). If you specify DU as the device, RESORC lists the device's unit, port, and partition settings.

The following sample shows the table that RESORC prints when you use the /D option.

```
* /D

DEVICE     STATUS          CSR      VECTOR(S)

  DY       122620          177170   264
  DD       Installed       176500   300 304
  DL       Installed       174400   160
  DX       Not installed   177170   264
  LS       Installed       176500   300 304
  LP       Installed       177514   200
  MS       Installed       172522   224 300
  DU       Installed       172150   154
  NL       Installed       000000   000
  LD       Installed       000000   000
  DM       Installed       177440   210
  VM       Installed       177572   000
  RK       Resident        177400   220
  SL       Not installed   000000   000
  MT       Installed       172520   224
  MM       Not installed   172440   224
```

In this table, the status column can list one of the following messages:

Message	Meaning
Installed	The device handler is in the system tables, but you have not loaded it in main memory (you can load it with the LOAD command).
Not installed	The device handler is not in the system tables, but you can add the handler with the INSTALL command (if there is a free slot).
-Not installed	The device handler special features do not match those of the monitor; you cannot install the handler. (The minus sign in front of any message means that you cannot install the handler.)
Resident	The device handler is permanently in memory; you cannot remove or unload it.
nnnnnn	The beginning address of a loaded handler.

The last column in the /D listing identifies vectors. If the handler has multiple vectors, the /D option prints the additional vectors in this column.

The next example shows handler status information for the device DU.

```
* DU:/D
DU0: is set UNIT=0,PART=0,PORT=0
DU1: is set UNIT=1,PART=0,PORT=0
DU2: is set UNIT=4,PART=0,PORT=1
DU3: is set UNIT=5,PART=0,PORT=1
DU4: is set UNIT=6,PART=0,PORT=1
DU5: is set UNIT=17,PART=2,PORT=1
DU6: is set UNIT=22,PART=0,PORT=0
DU7: is set UNIT=23,PART=0,PORT=0
```

14.2.4 Hardware Configuration Option (/H)

When you use the /H option, RESORC lists the processor type, the total amount of memory (in bytes) that the system contains, and all the special hardware features in your system configuration. The processor is one of the following:

LSI 11	PDP 11/24
PDT 130/150	PDP 11/34
PDP 11/04	PDP 11/35,40
PDP 11/05,10	PDP 11/44
PDP 11/15,20	PDP 11/45,50,55
SBC 11/21	PDP 11/60
PDP 11/23	PDP 11/70
PDP 11/23 PLUS	

Any special hardware is chosen from the following list. (The /H option displays the features in the order they occur in the list.)

FP11 Hardware Floating Point Unit
Commercial Instruction Set (CIS)
Extended Instruction Set (EIS)
Floating Point Instruction Set (FIS)
KT11 Memory Management Unit
Parity Memory
Cache Memory
VT11 or VS60 Graphics Hardware

The next item that appears in the /H listing is the clock frequency (50 or 60 cycles), and the last is the KW11-P programmable clock (if your system has one and you are not using it as the system clock).

The following example shows the /H option.

```
*. /H

PDP 11/23 Processor
60 KB of memory
FP11 Hardware Floating Point Unit
Extended Instruction Set (EIS)
KT11 Memory Management Unit
Parity Memory
60 Cycle System Clock
```

14.2.5 Loaded Jobs Option (/J)

The /J option prints information about the currently loaded jobs. For each job, RESORC displays:

1. The job number and name (if you have not enabled system job support on your monitor, the foreground job name appears as FORE, and its priority level is 1 in FB or XM)

2. The console the job is running on (with a nonmultiterminal monitor, this space is blank)

3. The priority level of the job
4. The job's state (running, suspended, or done but not unloaded)
5. The low and high memory limits of the job
6. The start address of the job's impure area

The following example uses the /J option.

```
*  /J

Job   Name   Console Level State     Low    High Impure
16    MFUNCT    1      7   Suspend 115350 127444 114412
14    EL        0      6           132404 141452 130663
12    QUEUE     0      5           152345 163243 144231
 0    RESORC    0      0   Run     000000 113144 133374
```

14.2.6 Device Assignments Option (/L)

When you type /L in response to the CSI asterisk, RESORC displays your system's device assignments. The devices RESORC lists are those in the system tables. The listing also includes additional information about particular devices. The informational messages and their meanings follow.

Message	Meaning
(RESORC) or =RESORC	The device or unit is assigned to the background job RESORC (for FB and XM monitors only).
(F) or =F	The device or unit is assigned to the foreground job (only for FB and XM monitors that do not have system job support).
(jobname) or =jobname	The device or unit is assigned to the system or foreground job (only for FB and XM monitors that have system job support), where jobname represents the name of the foreground or system job.
(Loaded)	The handler for the device has been loaded into memory with the LOAD command.
(Resident)	The handler for the device is included in the resident monitor.
=logical-device-name(1), logical-device-name(2)... ,logical-device-name(n)	The device or unit has been assigned the indicated logical device names with the ASSIGN command.
xx free slots	The last line tells the number of unassigned (free) devices.

The following example was created under an FB monitor. It shows the status of all devices known to the system.

```
* /L
TT   (Resident)
DL          (Resident)
            DL1 = SY , DK , OBJ, SRC, BIN
            DL2 = LST, MAP
MQ          (Resident)
RK
DM
DX          (Loaded)
            DX0: (F)
            DX1: (RESORC)
MT
LP
BA
NL
9 free slots
```

14.2.7 Current Monitor Option (/M)

When you use the /M option RESORC prints the type, version number, and update level of the currently running monitor. The designation BL, SJ, FB, or XM identifies the monitor type as base-line, single-job, foreground/background, or extended memory, respectively.

The following example uses the /M option.

```
* /M

RT-11FB (S) V05.00
```

14.2.8 Special Features Option (/O)

The special features chosen during system generation are listed on the terminal when you use the /O option. Whatever features are in effect are printed out in the same order as the following list of possible special features.

Option	Function
Device I/O timeout support	Permits device handlers to do the equivalent of a mark time without doing a .SYNCH request; DECnet applications require this support.
Error logging support	Keeps a statistical record of all I/O operations on devices that are supported by this feature; detects and stores data on any errors that occur during I/O operations.
Multiterminal support	Permits you to use as many as 16 terminals.
Memory parity support	Causes the system to print an error message when a memory parity error occurs.

SJ timer support	Configures the SJ monitor to include mark-time and cancel mark-time programmed requests and to support the .FORK process.
System job support	Allows you to run up to six jobs in the foreground in addition to the foreground and background jobs.

The following example shows the /O option.

```
* /O

Device I/O time-out support
Error logging support
Multi-terminal support
Memory parity support
```

If there are no special features in effect, RESORC prints NO SYSGEN options enabled.

14.2.9 Disk Subsetting Option (/S)

The /S option displays information about the subsetting of physical disks into logical disks. When you use the /S option, RESORC displays the logical disk unit with the file name to which it is assigned, and the size of the logical disk in decimal blocks.

The following example illustrates the /S option by showing the logical disks into which the physical disks DL0: and DL1: are divided.

```
* /S
LD0 is DL0:DISK.LST[4000.]
LD2 is DL1:DISK.SRC[1200.]*
LD1 is DL1:WORK.DSK[600.]
```

An asterisk (*) following the file information indicates that, although the logical disk assignment exists, the file does not exists on the volume that is currently mounted in the drive unit. A pound sign (#) indicates that the device handler is not loaded. These symbols are especially useful to determine the status of logical disk assignments after you use the SET LD CLEAN command or the LD /C option.

14.2.10 Terminal Status Option (/T)

The /T option displays information about currently running active terminals on multiterminal systems. Therefore, if your system does not include multiterminal support, RESORC prints:

```
No multi-terminal support
```

Since multiterminal support is not part of the distributed RT–11 monitors, such support is present on your system only if you have included it during system generation; that is, multiterminal support is a special feature.

If your system does include multiterminal support, and you enter the /T option in response to RESORC's asterisk, RESORC prints a table similar to the following:

```
* /T

Unit    Owner    Type          WIDTH TAB   CRLF  FORM   SCOPE   SPEED

0       RESORC   S-Console DL   132  No    Yes   No     No       N/A
1       FORE     Local     DZ    80  Yes   No    No     Yes     4800
```

Note that in this table, the unit number refers to the terminal; RT–11 multiterminal support permits you to use as many as 16 terminals.

The Unit column lists the terminal unit number, and the Owner column lists the name of the job (foreground, system, background, or none) to which the terminal is assigned. If the running monitor does not have system job support, RESORC prints FORE and RESORC, where applicable.

The Type column of this table shows the type of terminal – local, remote, console, or S-console (a console shared between background, system, and foreground jobs) – and the type of hardware interface the terminal uses – DL or DZ.

The WIDTH column indicates the width in characters (up to 255) of the terminal listing or display text.

The next four columns indicate which SET options are in effect on the terminal. If you have set TAB, the terminal can execute hardware tabs. If you have set CRLF, the terminal issues a carriage return and line feed whenever you attempt to type past the right margin. The terminal is capable of executing hardware form feeds if you have set FORM and, on graphics terminals, capable of echoing RUBOUT characters as backspace-space-backspace if you have set SCOPE.

The last column, SPEED, lists the terminal's baud rate (if interface is DZ). An N/A under the SPEED column indicates that the baud rate is not alterable (DL interface).

14.2.11 Physical Memory Layout Option (/X)

The /X option shows the organization of physical memory. The listing displays such information as where jobs are loaded, where the device handlers are loaded, where KMON and the USR reside, and the number of words of memory each occupies. Memory addresses are displayed in octal.

If you are running under the XM monitor, the listing is divided into two sections, the first for extended memory and the second for kernel memory.

The following example displays the organization of physical memory when running under the SJ monitor.

```
* /X

Address    Module     Words

160000     IOPAGE       4096.
157400     RK            120.
127274     RMON         6170.
126112     DY            313.
001000     ..BG..      21797.
```

The next example shows the organization of physical memory when running under the XM monitor.

```
* /X

     Extended Memory

Address    Module     Words

01000000   VM         393216.
00160000   ......     102400.


     Kernel Memory

Address    Module     Words

160000     IOPAGE      4096.
157350     RK           140.
124144     RMON        6970.
122612     DY           365.
111610     USR         2305.
001000     ..BG..     10620.
```

14.2.12 Summary Option (/Z)

The /Z option has the same effect as a combination of the /M, /C, /H, /J, and /O options. In other words, /Z lists the following information about your system:

- Monitor configuration
- Set options in effect on the monitor
- Hardware configuration
- System generation special features in effect

This information prints out in the order shown in the following sample.

```
* /Z

RT-11FB(S) V05.00
Booted from DL1:
```

```
USR is set SWAP
EXIT is set SWAP
KMON is set IND
TT is set NOQUIET
ERROR is set ERROR
KMON nesting depth is 3

PDP 11/23
60KB Memory
FP11 Hardware Floating Point Unit
Extended Instruction Set (EIS)
KT11 Memory Management Unit
Parity Memory
60 Cycle System Clock

Multi-terminal support
```

Chapter 15
Source Comparison (SRCCOM)

The RT–11 source comparison program (SRCCOM) compares two ASCII files and lists the differences between them. SRCCOM can either print the results or store them in a file. SRCCOM is particularly useful when you want to compare two similar versions of a source program. A file comparison listing highlights the changes made to a program during an editing session.

SRCCOM is also useful for creating a command file that you can run with the source language patch program (SLP), described in Chapter 21. When you use SRCCOM for creating a command file, you can patch one version of a source file so that it matches another version. Section 15.6 describes how to create a command file for SLP.

15.1 Calling and Terminating SRCCOM

To call SRCCOM from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

```
.R SRCCOM (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter a command string. If you respond to the asterisk by entering only a carriage return, SRCCOM prints its current version number.

You can type CTRL/C to halt SRCCOM and return control to the monitor when SRCCOM is waiting for input from the console terminal. You must type two CTRL/Cs to abort SRCCOM at any other time. To restart SRCCOM, type R SRCCOM or REENTER and a carriage return in response to the monitor's dot.

15.2 SRCCOM Command String Syntax

The syntax of the SRCCOM command string is:

[output-filespec],[SLP-filespec =]old-filespec,new-filespec[/option...]

where:

output-filespec	represents the destination device or file for the listing of differences.
SLP-filespec	represents the destination device or file for the command file to be run with SLP. See Section 15.6 for more information on creating a command file for SLP.
old-filespec	represents the first file to be compared.
new-filespec	represents the second file to be compared.
option	is one of the options listed in Table 15–1.

Note that you can specify the input files in any order if you want only a comparison. If you are creating a patch for use with the SLP utility, then specify the input files in the old-file, new-file order shown above. The console terminal is the default output device. The default file type for input files is .MAC. SRCCOM assigns .DIF as the default file type for differences files. The default file type for a SLP command file is .SLP.

Either or both output file specifications can be omitted. However the output file specifications are position-dependent. If you specify a SLP-filespec but no output-filespec, you must place a comma before the SLP file specification to denote the absence of an output-filespec.

SRCCOM examines the two source files line by line, looking for groups of lines that match. When SRCCOM finds a mismatch, it lists the lines from each file that are different. SRCCOM continues to list the differences until a specific number of lines from the first file matches the second file. The specific number of lines that constitutes a match is a variable that you can set with the /L:n option.

15.3 Using Wildcards with SRCCOM

You can use wildcards to perform multiple source file comparisons by typing only one command line. However, you can use wildcards only to compare files; you cannot use wildcards when creating a command file to run with SLP.

You can use wildcards in either input file specification (old-filespec or new-filespec). A different type of comparison is performed depending on whether you use wildcards in only one or in both of the input file specifications.

If you use wildcards in only one of the input file specifications, SRCCOM compares the file you specify without any wildcards to all variations of the file specification with wildcards. The wildcard represents the part of the file specification to be varied. You can use this method to compare one file to several other files. For example, when the following command line is executed, SRCCOM compares the file TEST1.MAC on device DY0: to all files on device DY1: with the file name TEST2:

```
*  TEST=DY0:TEST1.MAC,DY1:TEST2.*
```

You can send the results of all the comparisons to a file on a volume rather than to the console by specifying an output file. In this example, all differences from the comparisons are sent to the file TEST.DIF on device DK:.

If you use wildcards in both input file specifications, the wildcards represent a part of a file specification that you want to be the same in both files being compared. You can use this method to compare several pairs of files; each input file is compared to only one other input file. For example, when the following command line is executed, SRCCOM compares pairs of files; the first input file in each pair has the file name PROG1, and the second has the file name PROG2. The file type of both files in each pair must match.

```
*DY0:PROG1.*,DY1:PROG2.*
```

SRCCOM searches for the first file on DY0: with the file name PROG1, and takes note of its file type. Then, SRCCOM searches DY1: for a file with the file name PROG2 and the same file type as PROG1. If a match is found, SRCCOM compares the two files and lists the differences on the console (or sends the differences to an output file if one is specified). SRCCOM then searches DY0: for more files with the file name PROG1 and DY1: for PROG2 files with matching file types.

15.4 Options

Table 15–1 summarizes the operations you can perform with SRCCOM options. You can place these options anywhere in the command string, but it is conventional to place them at the end of the command line.

15.5 Differences Listing Format

This section describes the SRCCOM differences listing format and explains how to interpret it.

15.5.1 Sample Text

It will be helpful first to look at a sample text file, DEMO.BAK:

```
        FILE1
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHAME MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN:--
BELIEVE ME, HAPPINESS IS SLY,
  AND COMES NOT AYE WHEN SOUGHT, MAN.

                          --SCOTTISH SONG
```

Table 15–1: SRCCOM Options

Option	Function
/A	Lets you specify an audit trail (a string of characters that marks each updated line of a patched source file). Use /A with the SLP output file specification to create a file that can be used as command file input for the source language patch program SLP (see Chapter 21).
/B	Compares blank lines; normally, SRCCOM ignores blank lines.
/C	Ignores comments (all text on a line preceded by a semicolon) and spacing (spaces and tabs). A line consisting entirely of a comment is still included in the line count.
/D	Creates a listing of the new file specified in the command line with the differences from the old file marked with vertical bars (¦) to indicate insertions and bullets (o) to indicate deletions.
/F	Includes form feeds in the output listing; SRCCOM normally compares form feeds, but does not include them in the differences listing.
/L[:n]	Specifies the number of lines that determines a match; n is an octal integer in the range 1 through 310. The default value for n is 3.
/S	Ignores spaces and tabs.
/T	Compares blanks and tabs that appear at the end of a line. Normally SRCCOM ignores these trailing blanks and tabs.
/V:i:d	Used with /D to specify the characters you want SRCCOM to use in place of vertical bars and bullets. This option is useful if your terminal does not print the vertical bar character. Both i and d represent the numeric codes for ASCII characters in the range 40 through 176 (octal), where i represents the code for the insertion character and d the deletion character code.

This file contains two typing errors. In the fourth line of the song, shame should be share. In the seventh line, sly should be shy. Here is a file called DEMO.TXT that has the correct text:

```
        FILE2
HERE'S A BOTTLE AND AN HONEST FRIEND!
  WHAT WAD YE WISH FOR MAIR, MAN?
WHA KENS, BEFORE HIS LIFE MAY END,
  WHAT HIS SHARE MAY BE O' CARE, MAN?
THEN CATCH THE MOMENTS AS THEY FLY,
  AND USE THEM AS YE OUGHT, MAN:--
BELIEVE ME, HAPPINESS IS SHY,
  AND COMES NOT AYE WHEN SOUGHT, MAN.

                          --SCOTTISH SONG
```

15.5.2 Sample Differences Listing

SRCCOM can list the differences between the two files. The example below compares the original file, DEMO.BAK, to its edited version, DEMO.TXT:

```
_DEMO.BAK,DEMO.TXT/L:1
1) DK:DEMO.BAK
2) DK:DEMO.TXT
**********
1)1                FILE1
1)        HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1                FILE2
2)        HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)        THEN CATCH THE MOMENTS AS THEY FLY,
****
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)        THEN CATCH THE MOMENTS AS THEY FLY,
**********
1)1       BELIEVE ME, HAPPINESS IS SLY,
1)          AND COMES NOT AYE WHEN SOUGHT, MAN.
****
2)1       BELIEVE ME, HAPPINESS IS SHY,
2)          AND COMES NOT AYE WHEN SOUGHT, MAN.
**********
?SRCCOM-W-Files are different
```

If the files are different, SRCCOM always prints the file name of each file as identification:

```
1) DK:DEMO.BAK
2) DK:DEMO.TXT
```

The numbers at the left margin have the form n)m, where n represents the source file (either 1 or 2) and m represents the page (delineated by form feeds) of that file on which the specific line is located.

SRCCOM next prints ten asterisks and then lists the differences between the two files. The /L:n option was used in this example to set to 1 the number of lines that must agree to constitute a match.

The first line of both files differs. SRCCOM prints the first line from the first file, followed by the second line as a reference. SRCCOM then prints four asterisks, followed by the corresponding two lines of the second file.

```
1)1                FILE1
1)        HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1                FILE2
2)        HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
```

The fourth line contains the second discrepancy. SRCCOM prints the fourth line from the first file, followed by the next matching line as a reference.

```
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)        THEN CATCH THE MOMENTS AS THEY FLY,
****
```

The four asterisks terminate the differences from the first file. SRCCOM then prints the fourth line from the second file, again followed by the next matching line as a reference:

```
2)1          WHAT HIS SHARE MAY BE O' CARE, MAN?
2)         THEN CATCH THE MOMENTS AS THEY FLY,
**********
```

The ten asterisks terminate the listing for a particular differences section.

SRCCOM scans the remaining lines in the files in the same manner. When it reaches the end of each file, it prints the ?SRCCOM-W-Files are different message on the terminal.

The following example is slightly different. The default value for the /L:n option sets to 3 the number of lines that must agree to constitute a match. The output listing is directed to the file DIFF.TXT on device DK:.

```
* DIFF.TXT=DEMO.BAK,DEMO.TXT
?SRCCOM-W-Files are different
```

The monitor TYPE command lists the information contained in the output file:

```
.TYPE DIFF.TXT
1) DK:DEMO.BAK
2) DK:DEMO.TXT
**********
1)1                FILE1
1)       HERE'S A BOTTLE AND AN HONEST FRIEND!
****
2)1                FILE2
2)       HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
1)1        WHAT HIS SHAME MAY BE O' CARE, MAN?
1)       THEN CATCH THE MOMENTS AS THEY FLY,
1)         AND USE THEM AS YE OUGHT, MAN:--
1)       BELIEVE ME, HAPPINESS IS SLY,
1)         AND COMES NOT AYE WHEN SOUGHT, MAN.
****
2)1        WHAT HIS SHARE MAY BE O' CARE, MAN?
2)       THEN CATCH THE MOMENTS AS THEY FLY,
2)         AND USE THEM AS YE OUGHT, MAN:--
2)       BELIEVE ME, HAPPINESS IS SHY,
2)         AND COMES NOT AYE WHEN SOUGHT, MAN.
**********
```

As in the first example, SRCCOM prints the file name of each file:

```
1) DK:DEMO.BAK
2) DK:DEMO.TXT
```

The first line of both files differs, so SRCCOM prints the first two lines of both files, as in the listing at the terminal from the previous example:

```
1)1                FILE1
1)       HERE'S A BOTTLE AND AN HONEST FRIEND!
****
```

```
2)1                FILE2
2)      HERE'S A BOTTLE AND AN HONEST FRIEND!
**********
```

Again, the fourth line differs. SRCCOM prints the fourth line of the first file, followed by the next matching line:

```
1)1          WHAT HIS SHAME MAY BE O' CARE, MAN?
1)         THEN CATCH THE MOMENTS AS THEY FLY,
```

However, SRCCOM did not find a match (three identical lines) before it encountered the next difference. So, the second matching line prints, followed by the next differing line from the first file:

```
1)           AND USE THEM AS YE OUGHT, MAN:--
1)         BELIEVE ME, HAPPINESS IS SLY,
```

Again, the next matching line prints:

```
1)           AND COMES NOT AYE WHEN SOUGHT, MAN.
```

The /B option to include blank lines in the comparison was not used in this example. Thus, SRCCOM recognizes only one more line before the end of file. Since the two identical lines do not constitute a match (three are needed), SRCCOM prints the last line as part of the differences for the first file:

```
1)
1)                                --SCOTTISH SONG
1)
****
```

In a similar manner, SRCCOM prints the differences for the second file, ending the listing with the ?SRCCOM-W-Files are different message.

NOTE

> Regardless of the output specification, the differences message always prints on the terminal. If you compare two files that are identical and specify a file for the differences listing, the message ?SRCCOM-I-No differences found prints on the terminal and SRCCOM does not create an output file.

15.5.3 Changebar Option (/D[/V:i:d])

When you use the /D option in the SRCCOM command line, SRCCOM creates a listing in which it inserts vertical bars (`¦`) and bullets (`o`) to denote the differences between the two files in the command line. The vertical bar indicates insertion; the bullet indicates deletion. If you do not specify an output file, SRCCOM prints the listing at the terminal.

If you include the /V:i:d option with /D (you cannot use /V:i:d without /D), you can specify what characters you would like in place of the vertical bar and/or bullet. The argument i represents the ASCII code (between 40 and 176 octal) for the character you want in place of the vertical bar. The argument d represents the ASCII code (between 40 and 176 octal) for the character you want to use in place of the bullet.

In the following command line, SRCCOM compares DEMO.BAK to DEMO.TXT:

```
* DEMO.BAK,DEMO.TXT/D/L:1
```

When SRCCOM processes the last command, it prints at the terminal the following listing:

```
!                    FILE2
            HERE'S A BOTTLE AND AN HONEST FRIEND!
              WHAT WAD YE WISH FOR MAIR, MAN?
            WHA KENS, BEFORE HIS LIFE MAY END,
!             WHAT HIS SHARE MAY BE O' CARE, MAN?
            THEN CATCH THE MOMENTS AS THEY FLY,
              AND USE THEM AS YE OUGHT, MAN:--
!           BELIEVE ME, HAPPINESS IS SHY,
              AND COMES NOT AYE WHEN SOUGHT, MAN.
                                    --SCOTTISH SONG
    ?SRCCOM-W-Files are different
```

15.6 Creating a SLP Command File

You can use SRCCOM to create an input command file to the source language patch program, SLP, described in Chapter 21. Specify a SLP file specification in the SRCCOM command line. The SRCCOM option /A can be used for specifying an audit trail while creating this command file.

When you specify a SLP-filespec file in the SRCCOM command line, SRCCOM creates a file you can use as the SLP input command file. If you specify both an output-filespec and a SLP-filespec, SRCCOM creates both a differences listing and a SLP input command file. If you specify only an output-filespec, SRCCOM generates only a differences listing. If you specify only a SLP-filespec, SRCCOM creates only a SLP input command file.

In the following sample command line, SRCCOM creates an output file, MOD.SLP, which contains the necessary commands that, when used with SLP, can modify DEMO.BAK so that it matches DEMO.TXT.

```
* ,MOD=DEMO.BAK,DEMO.TXT
```

You can use the /A option to specify an audit trail. When SLP updates a file, it creates two output files. One output file is the patched source file; the second is a listing file. The listing file contains a numbered listing of the

patched file, and it also has an audit trail SLP has appended to each changed line. The audit trail is a string of characters that keeps track of the update status of each line in the patched source file. SLP appends the audit trail to the right margin of each updated line in the patched source file. Note that when SLP must change a line, it appends an additional audit trail below the audit trail of the changed line. The additional audit trail keeps track of the number of consecutive lines that change.

When you use the /A option you can specify what characters you want in the audit trail. SRCCOM prompts you for the audit trail:

```
Audit trail?
```

Respond with a string of up to 12 characters. Do not use a slash (/) in the audit trail. The example below provides a sample of a SLP listing file that contains a meaningful audit trail: the author's initials and the date of the patch.

```
SLP V05.00                    ADDRSS,ADDRSS=ADDRSS,ADDRSS

    1.   FOURSCORE AND SEVEN YEARS AGO,
    2.   OUR FATHERS BROUGHT FORTH ON THIS CONTINENT
    3.   A NEW NATION, CONCEIVED IN LIBERTY                ;AL-4*29*1863
    4.   AND DEDICATED TO THE PROPOSITION                  ;AL-4*29*1863
    5.   THAT ALL MEN ARE CREATED EQUAL.                   ;**-2
    6.   NOW WE ARE ENGAGED IN A GREAT CIVIL WAR,
    7.   TESTING WHETHER THAT NATION, OR ANY NATION        ;AL-4*29*1863
    8.   SO CONCEIVED AND SO DEDICATED,                    ;**-1
    9.   CAN LONG ENDURE.
```

Part II
System Jobs

Part II describes two utilities that you can run as system jobs: the Error Logger and the Queue Package. System job support is a special feature, available only through the system generation process. The Queue Package must run under an FB or XM monitor. The Error Logger runs under the SJ monitor as well as under the FB and XM monitors. For an in-depth description of the system job feature, see the *RT–11 Software Support Manual.*

Chapter 16 describes the error logging subsystem, which keeps statistical records of all I/O transfers, I/O errors, memory parity errors, and cache memory errors. Chapter 17 describes the Queue Package, which sends files for output to any valid RT–11 device.

Note that you can run these utilities as foreground jobs, but running them as system jobs enables you to run a foreground and a background job in addition.

Chapter 16
Error Logging Subsystem

The Error Logger monitors the hardware reliability of the system. The Error Logger keeps a statistical record of all I/O operations that occur on any of the following devices:

DD	DX
DL	DY
DM	RK
DU	

In addition to keeping these statistics, the Error Logger detects and records memory parity or cache errors and any errors that occur during I/O operations. At intervals you determine, the Error Logger produces individual and/or summary reports on some or all of these errors. The Error Logger is available only as a special feature: that is, you must perform the system generation process to enable Error Logging. It is available under the SJ monitor, the FB monitor, or the XM monitor. When you run the Error Logger with the FB or XM monitor, the Error Logger runs as either a foreground or system job.

16.1 Uses

Error logging reports are useful for maintaining the hardware on which RT–11 runs. Problems such as line noise, static discharges, or inherently error-prone media can cause recoverable errors on systems that are otherwise functioning normally. By studying error logging reports, you can learn to distinguish these errors from those that might be symptoms of an impending device failure. Also, some recoverable errors that are insignificant in themselves might be related to other more serious errors; their effects might not be immediately apparent to you. Information contained in the reports about each error and about the status of the system when the error occurred may alert you to a previously unforeseen hardware problem.

Sometimes a device fails so quickly that you are unable to prevent it. In this case, you can determine the cause more quickly if a report is available that describes the errors that occurred immediately prior to the failure.

In general, the error logging subsystem:

- Gathers device error and I/O transfer information from the handlers
- Gathers memory error information from the monitor

- Stores the information in a file or in an internal buffer
- Formats the information to produce a report

NOTE

Because the Error Logger can record data on each I/O transfer, thereby using additional computer time and memory, you may wish to use the Error Logger only when you experience difficulty with a device. Keeping a backup system volume on which the Error Logger is enabled makes this easy. You can also issue the command SET dd: NOSUCCES (dd represents the device mnemonic) before running the error logger. This command causes the device to call the Error Logger only when an I/O transfer fails. Successful I/O transfer statistics are not recorded. (Remember to reload the dd handler after issuing the SET dd: NOSUCCES command.)

16.2 Error Logging Subsystem

When used with the FB and XM monitors, the Error Logger consists of three programs and a statistics file. When you run the Error Logger, you coordinate these programs to gather I/O and error-related information into its statistics file and create the error report you want. The Error Logger names the statistics file it creates ERRLOG.DAT. At any time you specify, you call another Error Logger program, ERROUT, to create error reports from the information it has gathered in ERRLOG.DAT.

When used with the SJ monitor, the Error Logger uses only two programs, and ERRLOG.DAT is not created. Instead, the Error Logger gathers I/O and error-related information in an internal buffer area. You can then generate a report from the information in the internal buffer by calling a second Error Logger program, ERROUT.

The names and functions of the Error Logger programs follow

.EL.SYS — A pseudohandler used with the SJ monitor to gather information about errors that occur during I/O transfers. The device handlers detect success and error information as each I/O tranfer occurs. The handlers communicate this information to EL.SYS, which gathers all the necessary statistics for an error report. EL.SYS stores these statistics in an internal buffer whose default size is 1 block. You can change the size of the internal buffer by setting the conditional ERL$S (in SYCND.MAC) to n, where n is the number of blocks you want to reserve for the internal buffer. The variable n is interpreted as an octal number, unless you include a decimal point.

ERRLOG.REL A foreground or system job that gathers information about I/O transfers and system errors. The device handlers detect success and error information as each I/O transfer occurs. The handlers communicate this information to ERRLOG.REL, which stores all the necessary statistics for an error report in an internal buffer. The buffer's contents are transferred to ERRLOG.DAT periodically, and whenever you request an error report. When you initiate error logging with the FB or XM monitor, ERRLOG.REL instructs you to start up the second error logging program, ELINIT.

ELINIT A background job under the FB or XM monitor that creates and maintains the statistics file, ERRLOG.DAT. You can direct ELINIT to initialize ERRLOG.DAT every time you have a session at the terminal, or you can direct ELINIT to continue compiling statistics in ERRLOG.DAT on a daily basis.

When you run ELINIT, it prompts you for the information it needs to maintain ERRLOG.DAT's size. By default, ELINIT allocates 100 decimal blocks for ERRLOG.DAT. Each time you run ELINIT, it prints a message that tells how many of those 100 blocks are filled. If ERRLOG.DAT fills to its limit, EL.REL is unable to store more information in it. So that you can increase ERRLOG.DAT's size, ELINIT prompts you for a file size change each time you run the program.

If you bootstrap a monitor whose features differ from those of the monitor under which ERRLOG.DAT was created, ELINIT may print a message indicating that it must initialize ERRLOG.DAT to make the statistics it has been maintaining compatible with the new configuration. When this happens, ELINIT renames the ERRLOG.DAT it formerly maintained to ERRLOG.TMP and creates a new ERRLOG.DAT. The Error Logger can still create a report from ERRLOG.TMP.

Note that you do not use ELINIT when you run the Error Logger with the SJ monitor. Instead, the Error Logger compiles statistics in an internal buffer area. When the internal buffer area fills to its limit EL is unable to store more information in it. You can generate a report from the information in the internal buffer or purge the internal buffer at any time.

ERROUT A background job under the FB or XM monitor, or a program under the SJ monitor. ERROUT creates a report from the statistics in the EL internal buffer area, or from

ERRLOG.DAT or any file of that format. When you run ERROUT, you can direct the program to list the error report at the terminal or to create a file for the error report. You can also indicate whether you want a detailed report on each error that occurred or simply a summary report.

Figure 16–1 provides a diagram of the error logging subsystem under the FB and XM monitors. Figure 16–2 provides a diagram of the error logging subsystem under the SJ monitor.

Figure 16–1: Error Logging Subsystem – FB and XM

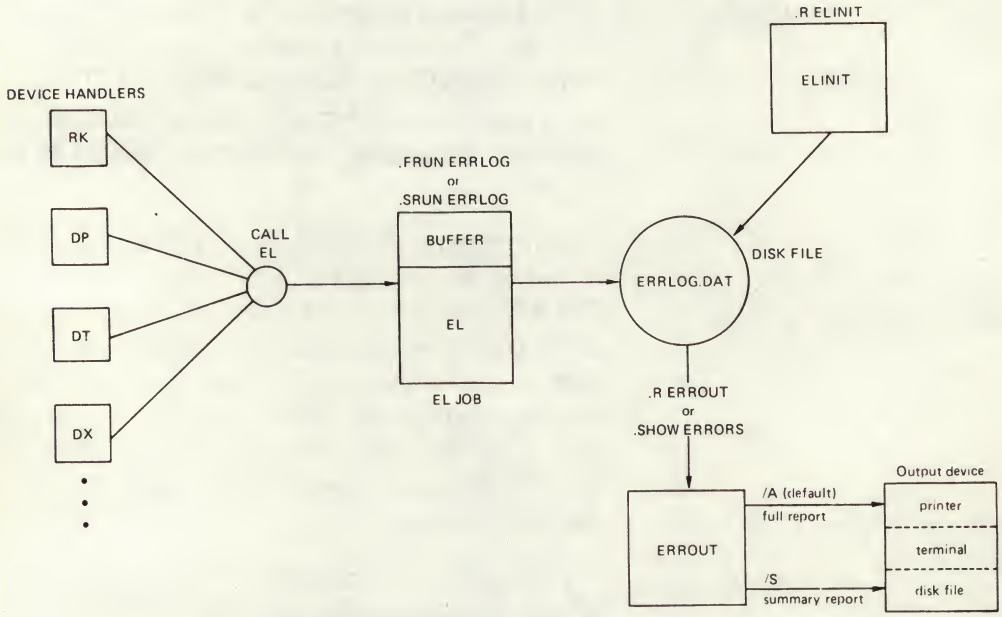
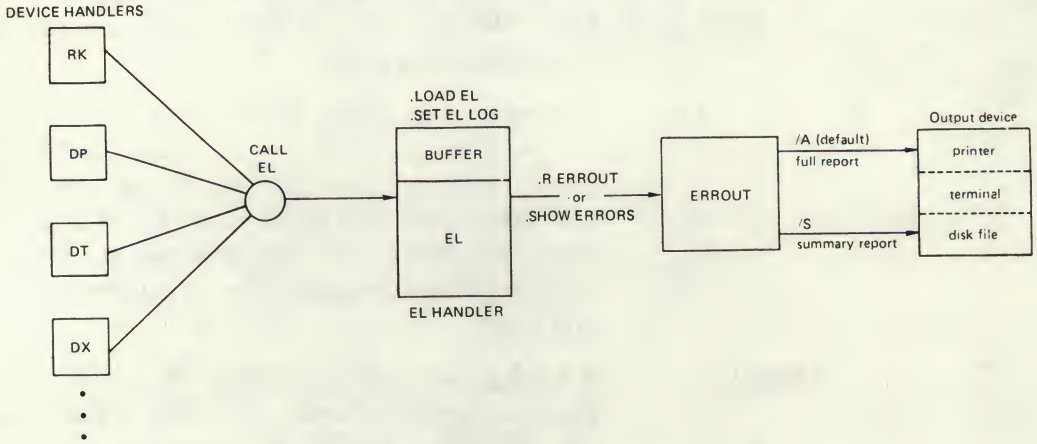


Figure 16–2: Error Logging Subsystem – SJ



16.3 Calling and Using the Error Logger with the SJ Monitor

To run the Error Logger with the SJ monitor, you must first load the Error Logger pseudohandler. Type in response to the keyboard monitor dot (.):

```
.LOAD EL RET
```

Then type the following command in response to the keyboard monitor dot (.) to enable error logging:

```
.SET EL LOG
```

When you type this command, the Error Logger begins to gather I/O transfer and error information in an internal buffer. The Error Logger also gathers statistics on the number of successful I/O transfers but does not create detailed records about successful transfers in the internal buffer. EL creates detailed records only for errors; these records contain such information as the device involved, when the error occured, register contents, and number of retries. If the buffer becomes full, EL continues to compile I/O transfer statistics but writes no further detailed records to the internal buffer. When this occurs, the Error Logger displays the following message:

```
?EL-W-Buffer is full, logging suspended
```

You can clear the contents of the internal buffer when it becomes full, or at any other time, by typing in response to the keyboard monitor dot (.):

```
. SET EL PURGE
```

This command clears only the detailed records on errors stored in the internal buffer; the I/O statistics are retained. Before you clear the contents of the buffer you can generate an error report. Section 16.5 describes how to generate and interpret a report.

To suspend error logging, type in response to the keyboard monitor dot (.):

```
. SET EL NOLOG
```

You can resume error logging by typing the SET EL LOG command.

You can disable error logging and unload the EL pseudohandler when you are through using the Error Logger by typing:

```
. UNLOAD EL
```

This command clears the EL internal buffer area and all I/O statistics as well. If you want to save the contents of the internal buffer, copy it to a file before you unload EL. To save the internal buffer contents, type a command with the following syntax in response to the keyboard monitor prompt:

```
. COPY EL: dev:filnam.typ
```

16.4 Calling and Using the Error Logger with the FB or XM Monitor

With the FB or XM monitors, the Error Logger runs only as a foreground or system job. To run the Error Logger as a foreground job, call the Error Logger from the system device by typing in response to the keyboard monitor dot (.):

```
. FRUN ERRLOG
```

To run the Error Logger as a system job, type in response to the keyboard monitor dot:

```
. SRUN ERRLOG
```

The Error Logger returns with a prompt, telling you how to initiate the error logging process.

```
?ERRLOG-I-To initiate Error Logging, RUN ELINIT
```

To terminate the Error Logger if it is running as a foreground job, type a CTRL/F followed by two CTRL/Cs. If it is running as a system job, type a CTRL/X and then specify ERRLOG as the system job you want to terminate (followed by two CTRL/Cs).

16.4.1 Using ELINIT

After you type RUN ELINIT (or R ELINIT) followed by a carriage return, ELINIT returns with a prompt. This prompt asks you to specify which device you want the statistics file ERRLOG.DAT written to. The format of this prompt follows.

```
What is the name of the device for the ERRLOG.DAT file <SY>?
```

Type a carriage return in response to the last prompt if you want ELINIT to write ERRLOG.DAT to the system device.

ELINIT then prints a message indicating how many blocks allocated to ERRLOG.DAT are in use. This message is followed by a prompt asking you if you want ELINIT to initialize ERRLOG.DAT. The format of the block usage message and the initialization prompt follows (where xx represents the number of blocks in use).

```
xx blocks currently in use of xx possible total in ERRLOG.DAT file

Do you want to zero the ERRLOG.DAT file and re-initialize (YES/NO) <NO>?
```

Type YES followed by a carriage return if you want ELINIT to initialize ERRLOG.DAT. When ELINIT initializes ERRLOG.DAT, it does not create a backup file for the statistics that were present prior to initialization. Enter a carriage return or type NO followed by a carriage return if you want ELINIT to retain the statistics already compiled in ERRLOG.DAT.

ELINIT proceeds by issuing the following prompt, asking you to indicate the number of blocks you want ELINIT to allocate to ERRLOG.DAT:

```
How many blocks for the ERRLOG.DAT file <nnn>?
```

The variable nnn represents the default size of 100, or the size of the current ERRLOG.DAT file. Type a carriage return if you want ERRLOG.DAT's file size to remain at the size indicated. If you want the file to be a different size, you can specify the number of blocks you want the file to have, followed by a carriage return. The only size limitation for ERRLOG.DAT is the amount of available space on the device in which it resides, and ERRLOG.DAT must be larger than one block.

NOTE

> Because of a rearrangement of your RT–11 configuration or bad header information in ERRLOG.DAT, it may be necessary for ELINIT to initialize ERRLOG.DAT even if you do not want it to. In this case, ELINIT automatically renames the current ERRLOG.DAT to ERRLOG.TMP, prints a message indicating it has done so, and returns the prompt How many blocks for the ERRLOG.DAT file <100>?.

After you have responded to the file size prompt, ELINIT prints the following message:

```
RT-11 V5.0 ERROR LOGGING INITIATED
```

After the Error Logger has printed the last message, you can proceed.

16.5 Using ERROUT

The Error Logger program, ERROUT, creates a report from the information compiled in the file ERRLOG.DAT or in EL's internal buffer. You can instruct ERROUT to generate a report either indirectly, by typing the SHOW ERRORS command, or directly by running ERROUT. See Chapter 4 of the *RT–11 System User's Guide* for more information on the SHOW ERRORS command. To call ERROUT directly from the system device, type in response to the keyboard monitor dot (.):

```
.RUN ERROUT
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal and waits for you to enter a command string according to the following general syntax:

[output-filespec =][input-filespec]/option

where:

output-filespec — represents the device to which you want ERROUT to type the report. If you do not specify an output device, ERROUT prints the report at the terminal. If you specify a file name, ERROUT writes the error report to that file.

input-filespec — represents ERRLOG.DAT or any file of the Error Logger statistics file format. (Thus, you can rename ERRLOG.DAT at any time and save it for later report formatting.) If you do not specify an input file, ERROUT assumes ERRLOG.DAT when running under the FB or XM monitor, and EL.SYS's internal buffer area when running under the SJ monitor.

option — is one of the options listed below.

/A — creates a report on each error in addition to a summary report of the errors and I/O transfers that occurred with each device.

/F:date — use to create an error report for errors logged from the date you specify. Specify the date in the form dd:mmm:yy, where dd represents the two-digit day, mmm represents the first three letters of the month, and yy represents the last two digits of the year. ERROUT interprets the date you enter as octal; use a decimal point with the day and year to indicate the date is in decimal. If you do not use /F:date, ERROUT creates a report starting with the first error logged in the work file.

/S — creates only a summary report of the errors and I/O transfers that occurred with each device.

/T:date — use to create an error report for errors logged up to the date you specify. Specify the date as with the /F:date option above. If you do not use /T:date, ERROUT creates a report that includes the last error logged in the work file.

If you enter only a carriage return in response to the CSI asterisk, ERROUT types a full report from ERRLOG.DAT at the terminal.

16.6 Report Analysis

This section provides a line-by-line analysis of each different report the Error Logger creates. Basically, there are three report categories:

- Storage device error report
- Memory error report
- Summary report

16.6.1 Storage Device Error Report

When a device handler encounters an error during an I/O transfer, it automatically retries that transfer as many as eight times (the actual number of times a handler retries an unsuccessful transfer depends on the particular device handler and on the value you specify for n with the SET dd: RETRY = N command). Regardless of the number of retries, each unsuccessful transfer will be recorded as only one entry in the error report, unless the registers change during the retries. In that case, the Error Logger creates a report for each retry.

Figure 16–3 provides an example of a storage device error report. This example is a report of the second attempt for a read operation on an RX02 double-density diskette. Table 16–1 tells what some of the lines in the report mean. For ease of reference, each line in this example report is numbered (although lines in the actual report are not numbered).

Figure 16–3: Sample Storage Device Error Report

```
1  ******************************************************************
2  DISK DEVICE ERROR
3  LOGGED 8-OCT-82 16:12:45
4  ******************************************************************
5
6  UNIT IDENTIFICATION
7          PHYSICAL UNIT NUMBER                                 000001
8          TYPE                                             RX211/RX02
9
10 SOFTWARE STATUS INFORMATION:
11         MAXIMUM RETRIES        8.
12         REMAINING RETRIES      6.
13         OCCURRENCES OF THIS ERROR WITH IDENTICAL REGISTERS  2.
14
15 DEVICE INFORMATION
16         REGISTERS:
17         RX2CS              114560
18         RX2DB              010400
19         RX2ES              000120
20
21         ACTIVE FUNCTION                    READ
22         BLOCK                            000001
23         PHYSICAL BUFFER ADDRESS START    003734
24         TRANSFER SIZE IN BYTES              512.
```

Table 16–1 explains each line in the sample report shown in Figure 16–3.

Table 16–1: Line-by-Line Analysis of the Sample Storage Device Error Report

Line	Explanation
1–4	Report header. Includes the date and time error was logged.
6–8	Unit identification. Identifies the drive number, the device controller, and the storage device type.
10–13	Retry count. Line 11 shows the maximum number of retries the device handler can perform. Line 12 tells the number of retries left before the transfer fails. If the number of remaining retries is 0, the transfer has failed. If the number of remaining retries is not 0, this usually indicates that a soft error has occurred, or that the transfer failed and the registers differed. In this example, with 6 retries remaining, the report was generated on the second retry. Line 13 tells how many times the error occurred with the same register contents.
15	Labels the section on device information. The lines that follow provide statistics on the device registers and address information.
16–19	Register contents. Each device has a number of hardware registers, the contents of which are listed in these lines.
21	I/O transfer type. Tells whether the I/O transfer was a read or write operation.
22	Device block number. Tells which device block the error occurred in.
23	Physical buffer start address. Tells the physical address in memory of the user data buffer for this I/O transfer.
24	Transfer size in bytes. Tells the size in bytes of the unit of data the device handler has attempted to transfer.

16.6.2 Memory Error Report

There are two kinds of memory errors for which the Error Logger creates reports: memory parity errors and cache memory errors. Figure 16–4 provides an example of a memory parity error report. As with the storage device report, this listing is numbered in the manual to aid in describing its contents. The listings that you obtain do not have line numbers.

Figure 16–4: Sample Memory Parity Error Report

```
1   ******************************************************************
2   MEMORY PARITY ERROR
3   LOGGED 8-OCT-82 16:13:22
4   ******************************************************************
5
6   SOFTWARE STATUS INFORMATION:
7           SYSTEM REGISTERS:
8           PC   001026
9           PSW  000000
10          OCCURRENCES OF THIS ERROR WITH IDENTICAL PC  3.
11
```

(Continued on next page)

```
12 DEVICE INFORMATION
13          MEMORY REGISTERS:
14          ADDRESS       CONTENTS
15          172100        100001
16
17          MEMORY SYSTEM ERROR REGISTER:     100000
18          CACHE CONTROL REGISTER:           000000
19          HIT/MISS REGISTER:                027000
20
21          ERROR TYPE IS MEMORY
```

Table 16–2 tells what each line in the last report shown in Figure 16–4 means.

Table 16–2: Line-by-Line Analysis of the Sample Memory Error Report

Line	Explanation
1–4	Report header. Tells the date and time the error was logged.
7–10	System register contents. Gives the contents of the program counter and the processor status word at the time of the error, as well as the number of times the program counter was the same for this error.
13–15	Memory parity register contents. Identifies your system's memory parity control and status register(s) and gives their contents.
17–19	Cache memory register contents. This information is displayed for both a memory parity error and a cache memory error if your system includes cache memory. See the *PDP–11 Processor Handbook* for more information on the cache memory registers.
21	Error type. Tells whether the error was a memory error or a cache memory error (see the following cache memory report for cache memory statistics).

The report in Figure 16–5 is an example of the report the Error Logger creates when it logs a cache memory error.

Figure 16–5: Sample Cache Memory Error Report

```
1  ******************************************************************
2  CACHE MEMORY ERROR
3  LOGGED 8-OCT-82 16:21:20
4  ******************************************************************
5
6  SOFTWARE STATUS INFORMATION:
7          SYSTEM REGISTERS:
8          PC   001026
9          PSW  000000
10         OCCURRENCES OF THIS ERROR WITH IDENTICAL PC  3.
11
12 DEVICE INFORMATION
13         MEMORY REGISTERS:
14         ADDRESS       CONTENTS
```

(Continued on next page)

```
15          172100        100001
16
17          MEMORY SYSTEM ERROR REGISTER:     000200
18          CACHE CONTROL REGISTER:           000000
19          HIT/MISS REGISTER:                000032
20
21          ERROR TYPE IS CACHE
```

The description provided in Table 16–2 also applies to Figure 16–5. Line 21 indicates that the memory error was in cache memory.

16.6.3 Summary Error Report

The summary error report provides statistics for all the devices the Error Logger supports. These statistics include counts for successful and unsuccessful I/O transfers for storage devices, and error counts for memory errors. The report consists of three sections:

- Device statistics
- Memory statistics
- Report file environment and error count

Figure 16–6 provides an example of a summary error report.

Figure 16–6: Sample Summary Error Report for Device Statistics

```
1   ******************************************************************
2   DEVICE STATISTICS
3   LOGGED SINCE 8-OCT-82 16:01:12
4   ******************************************************************
5
6   UNIT IDENTIFICATION
7          PHYSICAL UNIT NUMBER                                 000000
8          TYPE                                       RL11/RL02/RL02
9
10  DEVICE STATISTICS FOR THIS UNIT:
11         NUMBER OF ERRORS LOGGED                 0.
12         NUMBER OF ERROR RECEIVED                0.
13         NUMBER OF READ SUCCESSES               65.
14         NUMBER OF WRITE SUCCESSES               4.
15
16  UNIT IDENTIFICATION
17         PHYSICAL UNIT NUMBER                                 000000
18         TYPE                                           RX211/RX02
19
20  DEVICE STATISTICS FOR THIS UNIT:
21         NUMBER OF ERRORS LOGGED                 1.
22         NUMBER OF ERRORS RECEIVED               1.
23         NUMBER OF READ SUCCESSES                0.
24         NUMBER OF WRITE SUCCESSES               0.
25
26  UNIT IDENTIFICATION
27         PHYSICAL UNIT NUMBER                                 000001
28         TYPE                                           RX211/RX02
29
30  DEVICE STATISTICS FOR THIS UNIT:
31         NUMBER OF ERRORS LOGGED                 0.
32         NUMBER OF ERRORS RECEIVED               0.
33         NUMBER OF READ SUCCESSES                2.
34         NUMBER OF WRITE SUCCESSES               0.
```

The Error Logger provides summary statistics for each device. Notice that for each device, the count of the number of errors logged and the count of the number of errors received can be different. Sometimes, the Error Logger may receive an error but be unable to log it. This is usually due to full buffers or some other momentary software limitation. However, even if the Error Logger is unable to log an error, it is at least able to inform you of this fact.

Figure 16–7 provides an example of the second section of the summary error report, memory statistics. This report immediately follows the report on device statistics.

Figure 16–7: Sample Summary Error Report for Memory Statistics

```
1  ****************************************************************
2  MEMORY STATISTICS
3  LOGGED SINCE 8-OCT-82 16:01:12
4  ****************************************************************
5
6  STATISTICS:
7          NUMBER OF MEMORY PARITY ERRORS                    3.
8          NUMBER OF CACHE ERRORS                            0.
```

Figure 16–8 provides an example of the third section of the error report summary, the report file environment and error count.

Figure 16–8: Sample Report File Environment and Error Count Report

```
1  REPORT FILE ENVIRONMENT:
2          INPUT FILE                    DLO:ERRLOG.DAT
3          OUTPUT FILE                   LP :      .LST
4          OPTIONS                                 /A
5          DATE INITIALIZED                   8-OCT-82
6          DATE OF LAST ENTRY                 8-OCT-82
7
8  TOTAL ERRORS LOGGED                                  15.
9  MISSED REPORTS (TASK NOT READY)                      11.
10 MISSED REPORTS (BUFFER FULL)                          0.
11 MISSED REPORTS (FILE FULL)                            0.
12 UNKNOWN DEVICE STATISTICS ENTRIES                     0.
13 UNKNOWN ERROR RECORD ENTRIES                          0.
```

The segment of the report file environment shown in Figure 16–8 provides information concerning the input report file name (usually ERRLOG.DAT or ERRLOG.TMP) and the output report file name (any name that you specify in the initial ERROUT command line). In line 5, the report tells when the input report file was initialized, and in line 6, the date of the last error entry to the input report file.

Lines 8 through 13 count additional error count statistics. Lines 9 through 11 count the number of missed reports. A missed report is an I/O transfer or error for which the Error Logger was unable to gather information because ERRLOG was running but ELINIT had not been run, the internal buffer was full, or the ERRLOG.DAT statistics file was full.

Line 12 provides a count of unknown device statistics entries. An unknown device statistics entry occurs when ERROUT does not recognize the device identifier byte the EL program recorded in the statistics portion of the ERRLOG.DAT file. (All DIGITAL-distributed device handlers that support Error Logging can be identified by ERROUT, so this problem occurs most often with user-written handlers. See the *RT–11 Software Support Manual* for details on adding a device to ERROUT.)

Line 13 keeps a count of the unknown error record entries. This condition occurs when the ERROUT task cannot identify a device error recorded in the ERRLOG.DAT file. (Again, this condition occurs most often with user-written handlers.)

Chapter 17
Queue Package

The Queue Package is a utility you can use for sending files to any valid RT–11 device. Although the Queue Package is particularly useful for queuing files for printing, queuing is not restricted to a line printer or any other serial device.

The Queue Package consists of two programs and a work file that contains the lineup of files, or queue, waiting to be output:

QUEUE	Queues and sends the files you specify. QUEUE runs as a foreground or system job.
QUEMAN	A background job that processes command lines and file specifications you enter, and sends that information to QUEUE. It serves as the interface between you and the Queue Package.
QUFILE.WRK	Contains the queue for the files QUEUE sends to the device(s) you specify.

The Queue Package runs only with the FB or XM monitor.

NOTE

To prevent QUEUE and another job from intermixing output on the same non-file-structured device, use the LOAD command to assign exclusive ownership of a device to QUEUE.

17.1 Calling and Using the Queue Package

To use the Queue Package, you must first run QUEUE from the system volume as either a foreground or system job. (Note that system job support is a special feature. You can perform the system generation process to build a monitor and handlers that support system jobs.) You can then run QUEMAN in the background when you are ready to output files.

17.1.1 Running QUEUE

To run QUEUE as a foreground job, call QUEUE from the system volume by typing in response to the keyboard monitor dot (.):

```
.FRUN QUEUE (RET)
```

To run QUEUE as a system job, type in response to the keyboard monitor dot:

```
.SRUN QUEUE (RET)
```

To halt QUEUE, see Section 17.2.1.

17.1.2 Running QUEMAN

To run QUEMAN from the system volume, type in response to the keyboard monitor dot (.):

```
.R QUEMAN
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the terminal, indicating it is ready to accept input. Enter a command string according to this general syntax:

[dev:[jobname[/options]] =][filespec][/options][,filespec[/options]...]

where:

dev:	represents any valid RT–11 device. (The default output device is LP0:.)
jobname	represents the output job name. This is the logical name for all the files specified in the command. If you send a job to a file-structured device, QUEUE uses this name as the file name of the job, and assigns a .JOB file type. If you do not specify a job name, QUEMAN uses the file name of the first input file. The job name can have up to six characters.
filespec	represents the input file specification. If you do not specify a file type, QUEMAN assumes a .LST file type.
options	represents one or more of the options from Table 17–1.

If you use commas in place of file specifications, QUEMAN ignores all remaining file specifications on the command line. (Note, however, that if your command string consists of several lines, entering commas in place of a file specification does not affect file specifications on subsequent lines in the command string. Using commas in place of a file specification affects only those remaining files in that particular command line.)

17.2 QUEMAN Options

Table 17–1 summarizes the options you can use in the QUEMAN command line. The sections that follow Table 17–1 provide detailed explanations and examples of each option. Note that some of the options are position-dependent — that is, their function depends on where you place them in the

Table 17–1: QUEMAN Options

Option	Section	Function
/A	17.2.1	Terminates QUEUE.
/C[:date]	17.2.2	Prints only those files with the specified date. If you use /C and do not specify a date, QUEMAN prints only those files with the current date.
/D	17.2.3	Deletes the input file(s) after printing. This option is position-dependent.
/H:n	17.2.4	Prints n banner pages for each specified input file, where n is a decimal number. This option is position-dependent.
/I[:date]	17.2.5	Prints only those files created on or after the specified date.
/J[:date]	17.2.6	Prints only those files created before the specified date.
/K:n	17.2.7	Prints n copies of each specified file, where n is a decimal number. This option is position-dependent.
/L	17.2.8	Lists the contents of the queue.
/M	17.2.9	Removes a job from the queue.
/N	17.2.10	Specifies no banner pages for the input file(s).
/P	17.2.11	Sets two Queue Package default values: the number of banner pages, and whether you want QUFILE.WRK deleted when you terminate QUEUE.
/Q	17.2.12	Causes QUEMAN to request confirmation that a particular file should be included in the operation. QUEMAN prints the name of each file that can be included in the operation. You must respond Y to include a particular file.
/R	17.2.14	Resumes sending the current job after it has been suspended, or restarts the current file in the job being sent.
/S	17.2.13	Suspends output at the end of the current file.
/W	17.2.15	Prints on the console a log of the files involved in the operation.
/X	17.2.16	Allows QUEMAN to continue processing instead of halting when it cannot find a file you specified in the command line.
//	17.2.17	Continues command on the next line.

command line. Also, some of the options accept a date as an argument. The syntax for specifying the date is:

[:dd.][:mmm][:yy.]

where:

dd.	represents the day (a decimal integer in the range 1–31)
mmm	represents the first three characters of the name of the month
yy.	represents the year (a decimal integer in the range 73–99)

The default value for the date is the current system date. If you omit any of these values (dd, mmm, or yy), the system uses the values from the current system date. For example, if you specify only the year ::82. and the current system date is May 4, 1983, the system uses the date 4.:MAY:82.. If the current date is not set, it is considered 0 (the same as for an undated file in a directory listing). The date values are position-dependent. If you omit the day (dd) or month (mmm), you must use a colon (:) in place of the value.

If you have selected timer support through the system generation process, but have not selected automatic end-of-month date advancement, make sure that you set the date at the beginning of each month with the DATE command. If you fail to set the date at the beginning of each month, the system prints -BAD- in the creation date column of each file created beyond the end-of-month. (Note that you can eliminate -BAD- by using the RENAME/SETDATE command after you set the date.)

If QUEUE is sending a job that has multiple input files to an RT–11 file-structured volume, QUEUE copies each input file to a separate output file with the same file name and type as the input file. The jobname is used in the JOBNAME field of the banner page (if you request banner pages).

17.2.1 Terminating QUEUE (/A)

When you type /A in response to the CSI asterisk, QUEMAN terminates QUEUE. If you use /A while a job is printing, QUEUE halts output. If QUEUE is running as a foreground job, using /A has the same effect as typing CTRL/F and two CTRL/Cs. If QUEUE is running as a system job, using /A has the same effect as typing CTRL/X and then specifying QUEUE as the system job to which you want to direct input, followed by two CTRL/Cs.

The following example terminates QUEUE

```
.R QUEMAN
*/A
```

If you use CTRL/Cs to termiante QUEUE, this may take a few seconds because QUEUE performs the following I/O rundown before terminating:

- Waits for all current I/O transfers to complete
- Removes protection from the input file if it was unprotected before QUEUE began copying it to the output device
- Closes the work file if you have chosen to save the work file

17.2.2 Date Option (/C[:date])

The /C[:date] option prints only those files with the specified date. If no date is specified only those files with the current date are printed. Specify /C only once in the command line; it applies to all the file specifications in the entire

command. The following command prints on LP0: all files named ITEM1 and ITEM2 that also have the date March 20, 1983.

```
* ITEM1/C:20.:MAR:83.,ITEM2
```

17.2.3 Deleting Input Files After Printing (/D)

Use the /D option to delete input files after QUEUE has sent them. This option is position-dependent. If you use it with the job name, /D applies to all the input files. If you use it with an input file specification, /D applies only to that input file.

The following example deletes all input files after they have been sent:

```
* MYJOB/D=FILE1,FILE2,FILE3
```

The following example deletes FILE1 and FILE3 but retains FILE2 after QUEUE has sent them:

```
* MYJOB=FILE1/D,FILE2,FILE3/D
```

Input files are protected from deletion while QUEUE is copying them to the output device. This protects input files from accidental deletion.

17.2.4 Printing Banner Pages (/H:n)

Use the /H:n option to print banner pages for the input files you specify, where n is a decimal number selecting the number of banner pages. This option is position-dependent. If you use /H:n with the jobname, QUEUE prints n banner pages for each input file. If you use /H:n with an input file specification, QUEUE prints n banner pages for that file, and prints the default number of banner pages for the remaining input files. (Note that you set the default number of banner pages with the /P option, described in Section 17.2.11.)

The sample command line that follows prints four banner pages for each input file:

```
* LAUGHN/H:4=ROWAN.TXT,MARTIN.TXT
```

The following sample command prints four banner pages for MARTIN.TXT and the default number of banner pages for ROWAN.TXT:

```
* LAUGHN=ROWAN.TXT,MARTIN.TXT/H:4
```

Note that QUEUE never prints a banner for the job; it prints banners only for the input files.

17.2.5 Since Option (/I[:date])

The /I[:date] option prints only those files created on or after the specified date. If you specify no date QUEMAN uses the current system date. The following command prints only those .MAC files on device DK: created on or after April 21, 1983:

```
* *.MAC/I:21.:APR:83.
```

17.2.6 Before Option (/J[:date])

The /J[:date] option copies only those files created before the specified date. If you specify no date QUEMAN uses the current system date. The following command prints only those .MAC files on device DK: created before April 21, 1983:

```
* *.MAC/J:21.:APR:83.
```

17.2.7 Printing Multiple Copies (/K:n)

Use the /K:n option to specify the number of copies of the input files you specify, where n is a decimal number. The /K:n option is position-dependent. If you use /K:n with the job name, QUEUE prints n copies of each input file. If you use /K:n with an input file specification, QUEUE prints n copies of that particular file.

The next command line prints four copies of LAUREL.LST and four copies of HARDY.LST:

```
* JOB/K:4=LAUREL,HARDY
```

The following sample command line prints four copies of LAUREL.LST and the default number of copies of HARDY.LST:

```
* JOB=HARDY,LAUREL/K:4
```

17.2.8 Listing the Contents of the Queue (/L)

Use the /L option to get a listing of the contents of the queue. The listing gives the output device, job name, input files, job status, and number of copies for each job that is in the queue. The job STATUS column prints P if the job is currently being sent, S if the job being sent is suspended, or Q if the job is waiting to be sent. If you have a large queue and your console is a video terminal, you can use the keyboard CTRL/S and CTRL/Q commands to control the scrolling of the listing.

The sample command line that follows lists the queue:

```
*
DEVICE     JOB     STATUS     COPIES  FILES

 LP0:      LAB2      P          1     PASS3 .LST
                                2     PASS4 .LST
                                2     PASS5 .LST
 LP0:      HODG      Q          3     MESMAN.DOC
 MT1:      JUDITH    Q          2     PART1 .DOC
                                2     PART2 .DOC
 LP0:      JOYCE     Q          1     SSM   .DOC
                                      DOCPLN.DOC
```

17.2.9 Removing a Job from the Queue (/M)

Use the /M option to remove a job from the queue. When you use this option, specify the job name followed by /M and the equal sign (=). The following example removes the job LAB4 from the queue:

```
* LAB4/M=
```

When you use /M, you do not have to specify the input files, only the job name. You remove all the files associated with the job name.

17.2.10 No Banner Pages Option (/N)

Use the /N option to specify that you do not want QUEUE to print any banner pages for the input file(s). Use /N if you have previously set the default number of banner pages with the /P option (see Section 17.2.11). The /N option is position-dependent; that is, if you use it after the job name, it applies to each input file. Use /N after an input file to apply to only the particular file.

The following example uses /N to specify no banner pages for each file in the job, MYJOB2.

```
* MYJOB2/N=PASS1,PASS2,PASS3
```

The /N option has the same effect as /H:0 (see Section 17.2.4).

17.2.11 Setting Queue Package Defaults (/P)

Use the /P option to set defaults for two values:

1. Number of banner pages printed for each input file. You can override the default number of banner pages by using the /H option.
2. Whether you want the work file, QUFILE.WRK, deleted when you halt QUEUE. (Note that QUFILE.WRK contains the lineup of files, or queue, waiting to be sent to an output device.)

When you type /P in response to the CSI asterisk, QUEMAN prints the following prompt at the terminal:

```
1) Number of banner pages ?
```

QUEUE uses the number you type as the default number of banner pages it prints for each file it sends to a device. If you type only a carriage return, QUEMAN assumes 0. This value remains in effect until the work file, QUFILE.WRK, is deleted (see below).

After you have responded to the previous prompt, QUEMAN prints the following prompt at the terminal:

```
2) Delete workfile ?
```

If you type N followed by a carriage return, or only a carriage return, QUEUE maintains the current QUFILE.WRK after you halt QUEUE. That is, if you start QUEUE later, QUFILE.WRK retains the queue it had prior to the halt. By maintaining QUFILE.WRK between the times QUEUE is halted, you have an automatic queue restart capability. This value remains in effect until you reset it.

If you type Y followed by a carriage return, QUEUE deletes QUFILE.WRK when you halt QUEUE. The next time you start QUEUE, it creates a new QUFILE.WRK. This value remains in effect only until the next time you start QUEUE.

17.2.12 Query Option (/Q)

Use the /Q option to list all files and to confirm individually which of these files should be printed. Typing Y or any string beginning with Y followed by a carriage return causes the named file to be printed. Typing anything else excludes the file. The following example prints files that reside on DY1:.

```
*DY1:*.*/Q
DY1:FIX463.SAV              to LP: ?
DY1:GRAPH.BAK               to LP: ?
DY1:DMPX.MAC                to LP: ?
DY1:MATCH.BAS               to LP: ?
DY1:EXAMP.FOR               to LP: ?
DY1:GRAPH.FOR               to LP: ?
DY1:GLOBAL.MAC              to LP: ?
DY1:PROSEC.MAC              to LP: ?
DY1:KB.MAC                  to LP: ?
DY1:EXAMP.MAC               to LP: ?
```

17.2.13 Suspending Output (/S)

Use the /S option to suspend output of a job being sent. When you type /S in response to the CSI asterisk, QUEUE suspends output only after it has completed output of the current file in the job. This option is useful if you want access to an output device while a large job is being sent to it.

To resume output, use the /R option (Section 17.2.14).

17.2.14 Resuming/Restarting Output (/R)

Use the /R option either to resume output of a suspended job, or to restart output of the current file in the job from the beginning of the file. Note that a job resumes if you previously suspended it with /S, and a job restarts if you have not previously suspended it.

Resuming a job with multiple input files when the job is being sent to an RT–11, file-structured volume can be useful if the volume involved is too small to contain the entire job. You can suspend the job being sent (using the /S option), change volumes, and resume output of the remainder of the job on the new volume. QUEUE uses the same file name for both parts of the job.

17.2.15 Log Option (/W)

When you use the /W option, QUEMAN prints a list of all files printed or copied to a file. The /W option is useful if you do not want to take the time to use the query mode (the /Q option, described in Section 17.2.12), but you do want a list of the files printed or copied by QUEMAN.

QUEMAN prints the log for an operation on the terminal under the command line. This example shows logging when files are queued to be printed on LP0:.

```
* DY1:*.*/W
Files queued:
DY1:TEST.MAC                    to LP:
DY1:FIX463.SAV                  to LP:
DY1:GRAPH.BAK                   to LP:
DY1:DMPX.MAC                    to LP:
DY1:MATCH.BAS                   to LP:
DY1:EXAMP.FOR                   to LP:
DY1:GRAPH.FOR                   to LP:
DY1:GLOBAL.MAC                  to LP:
DY1:PROSEC.MAC                  to LP:
DY1:EXAMP.MAC                   to LP:
```

17.2.16 Information Option (/X)

The /X option causes QUEMAN to print an information message when QUEMAN fails to find all of the files you specify in a command line. If you do not use /X, QUEMAN prints a fatal error message when it is unable to find an input file, and returns control to the keyboard monitor after the rest of the operation completes. Use /X in indirect command files to ensure that processing will continue even if QUEMAN fails to find a file you specify.

In the following example, QUEMAN is unable to find the file FILE2.MAC. QUEMAN prints a message informing you that the file was not found and continues processing.

```
* LP:*.*=DL0:FILE1.MAC,FILE2.MAC,FILE3.MAC
?QUEMAN-I-File not found DL0:FILE2.MAC
```

17.2.17 Continuing a Command String (//)

Use the // option to continue a command string on subsequent lines. This option is useful if you want to output more files than you can specify on one line. When you want to include several lines in a CSI command string, type // at the end of the first line, and again at the end of the command string.

The following command string uses the // option:

```
* JOBNAM=LML1.MAC,LML2A.MAC//
* LML51.MAC,LML95.MAC
* LML4.MAC,LML56.MAC//
```

Part III
Debugging and Altering Programs

Part III of this manual consists of the following four topics: on-line debugging technique (ODT), object module patch program (PAT), save image patch program (SIPP), and source language patch program (SLP). The four programs that these chapters describe can help you debug programs, examine or change assembled programs, and patch source programs.

Chapter 18 describes ODT. This program aids you in debugging assembly language programs. With ODT, you can control your program's execution, examine locations in memory and alter their contents, and search the object program for specific words.

Chapter 19 describes PAT, which patches or updates code in a relocatable binary object module. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module.

Chapter 20 describes SIPP. You can use SIPP to examine or modify individual locations within programs linked with the RT–11 V04 or later linker. Using SIPP, you can also create an indirect command file that contains a patch and the commands necessary to install it.

Chapter 21 describes SLP. SLP provides an easy way to make changes to source files. This program can accept an indirect command file created by the DIFFERENCES/SLP:filespec command (or by specifying a SLP-filespec to SRCCOM) to make two source programs match.

Chapter 18
On-Line Debugging Technique (ODT)

On-line debugging technique (ODT) is a program that aids in debugging assembly language programs. ODT performs the following tasks:

- Prints the contents of any location for examination or alteration
- Runs all or any portion of an object program using the breakpoint feature
- Searches the object program for specific bit patterns
- Searches the object program for words that reference a specific word
- Calculates offsets for relative addresses
- Fills a single word, block of words, byte, or block of bytes with a designated value

Make sure you have an assembly listing and a link map available for the program you want to debug with ODT. You can make minor corrections to the program on line during the debugging session, and you can then execute the program under the control of ODT to verify the corrections. If you need to make major changes, such as adding a missing subroutine, note them on the assembly listing and incorporate them in a new assembly.

See the *RT–11 Software Support Manual* for debugging the following routines and jobs: interrupt service routines, device handlers, multiterminal jobs, extended memory and virtual jobs.

18.1 Calling and Using ODT

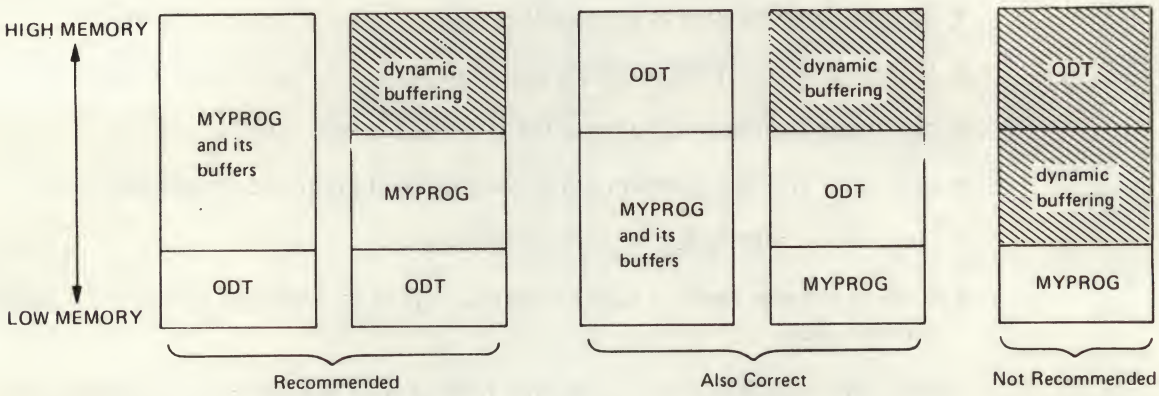
ODT is supplied as a relocatable object module. You can link ODT with your program (using the RT–11 linker) for an absolute area in memory and load it with your program. When you link ODT with your program, it is a good idea to link ODT low in memory relative to the program. If you do link ODT high in memory, be sure that the buffer space for your program is contained within program bounds. Otherwise, if your program uses dynamic buffering, program execution may destroy ODT in memory. Figure 18–1 shows the relationship between ODT and the program MYPROG in memory.

To link ODT low in memory relative to your program, the program must declare a named p-sect by using the .PSECT directive. Since the linker orders blank p-sects below named p-sects in memory, your program should declare a named p-sect so that ODT will be linked lower in memory than your program.

For example, if you include the directive .PSECT MYPROG in the program MYPROG, the following command will cause the linker to link ODT low in memory relative to MYPROG, and create the executable module MYPROG.SAV:

```
INK/MAP:TT/DEBUG MYPROG (RET)
```

Figure 18–1: Linking ODT with a Program



Once loaded in memory with your program, ODT has three legal start or restart addresses. Use the lowest (O.ODT) for normal entry, retaining the current breakpoints. The next (O.ODT + 2) is a restart address that clears all breakpoints and reinitializes ODT, thus saving the general registers and clearing the relocation registers. Use the last address (O.ODT + 4) to reenter ODT. A reenter saves the processor status and general registers, and removes the breakpoint instructions from your program. ODT prints the bad entry (BE) error message. Breakpoints that were set are reset by the next ;G command. (;P is invalid after a BE message.) The ;G and ;P commands control program execution and are explained in Section 18.3.7.

The system uses as an absolute address the address of the entry point O.ODT shown in the linker load map.

NOTE

If you link ODT with an overlay-structured file, it should reside in the root segment so that it will always be in memory. Remove all breakpoints from the current overlay segment before execution proceeds to another overlay segment. A breakpoint inserted in an overlay is destroyed if it is overlaid during program execution.

The following example links ODT low in memory relative to MYPROG, creating the executable module MYPROG.SAV. Running MYPROG causes ODT to start automatically.

```
.LINK/MAP:TT:/DEBUG MYPROG
RT-11 LINK  V08.00        Load Map          Thursday    04-Nov-82 14:15
Page 1
MYPROG            .SAV    Title:  ODT       Ident:  05.00

Section  Addr   Size     Global  Value    Global  Value    Global  Value

. ABS.   000000 001000 = 256.    words   (RW,I,GBL,ABS,OVR)
$ODT$    001000 006152 = 1589.   words   (RW,I,LCL,REL,CON)
                        O.ODT    001232
PROG     007152 002052 = 533.    words   (RW,I,LCL,REL,CON)
                        START    007152

Transfer address = 001232, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT  V05.00
*
```

The following example links MYPROG low in memory relative to ODT and specifies O.ODT as the transfer address. Running MYPROG causes ODT to start automatically. The advantage to this method is that MYPROG is loaded at its normal, execution-time address.

```
.,LINK/MAP:TT: MYPROG,ODT/TRANSFER
Transfer symbol? O.ODT

RT-11 LINK  V08.00        Load Map          Thursday    04-Nov-82 14:15
Page 1
MYPROG   .SAV             Title:  ODT  Ident:  05.00
Section  Addr   Size     Global  Value    Global  Value    Global   Value

. ABS.   000000 001000 = 256.    words   (RW,I,GBL,ABS,OVR)
PROG     001000 002052 = 533.    words   (RW,I,LCL,REL,CON)
                        START    001000
$ODT$    003052 006152 = 1589.   words   (RW,I,LCL,REL,CON)
                        O.ODT    003304

Transfer address = 003304, High limit = 011222 = 2377. words

.

.R MYPROG

 ODT V05.00
*
```

The following example is similar to the previous example, except that execution does not automatically begin with ODT. When you start the program (MYPROG in this case), you must specify the address of O.ODT as shown in the link map.

```
. LINK/MAP:TT: MYPROG,ODT
RT-11 LINK  V08.00        Load Map          Thursday    04-Nov-82 14:15
Page 1
MYPROG            .SAV    Title:  ODT       Ident:  05.00

Section   Addr    Size    Global   Value    Global   Value    Global   Value

. ABS.    000000 001000 = 256.    words   (RW,I,GBL,ABS,OVR)
PROG      001000 002052 = 533.    words   (RW,I,LCL,REL,CON)
                         START    001000
$ODT$     003052 006152 = 1589.   words   (RW,I,LCL,REL,CON)
                         O.ODT    003304

Transfer address = 003304, High limit = 011222 = 2377. words

. GET MYPROG

. START 3304

 ODT  V05.00
*
```

The next example links ODT with a bottom address of 4000, then loads ODT.SAV and MYPROG.SAV into memory. As in the example above, when you start the program, you must specify the address of O.ODT as shown in the link map.

```
. LINK/MAP:TT: ODT/BOTTOM:4000
RT-11 LINK  V08.00        Load Map          Thursday    04-Nov-82 14:15
Page 1
ODT       .SAV    Title:  ODT       Ident:  05.00       /B:004000

Section   Addr    Size    Global   Value    Global   Value    Global   Value

. ABS.    000000 004000 = 1024.   words   (RW,I,GBL,ABS,OVR)
$ODT$     004000 006152 = 1589.   words   (RW,I,LCL,REL,CON)
                         O.ODT    004232

Transfer address = 004232, High limit = 012150 = 2612. words

. GET ODT.SAV

. GET MYPROG.SAV

. START 004232

 ODT  V05.00
*
```

You can restart ODT by specifying O.ODT+2 as the start address. This reinitializes ODT and clears all breakpoints. For example:

```
. START 4234
*
```

You can reenter ODT by specifying O.ODT+4 as the start address. For example:

```
.START 4236

BE004242
*
```

If ODT is waiting for a command, a CTRL/C from the keyboard calls the keyboard monitor. The monitor responds with a ^C on the terminal and waits for a command. (You can use the REENTER command to reenter ODT only if your program has set the reenter bit and ODT is linked high in memory relative to the program; otherwise, ODT is reentered at address O.ODT + 6.)

If you type CTRL/U during a search printout, the search terminates and ODT prints an asterisk.

18.2 Relocation

When the assembler produces a relocatable object module, the base address of the module is assumed to be location 000000. The addresses of all program locations, as shown in the assembly listing, are relative to this base address. After you link the module, many of the values and all of the addresses in the program will be incremented by a constant whose value is the actual absolute base address of the module after it has been relocated. This constant is called the relocation bias for the module. Since a linked program may contain several relocated modules, each with its own relocation bias, and since, in the process of debugging, these biases will have to be subtracted from absolute addresses continually in order to relate relocated code to assembly listings, ODT provides automatic relocation.

The basis of automatic relocation is the eight relocation registers, numbered 0 through 7. You can set them to the values of the relocation biases at different times during debugging. Obtain relocation biases by consulting the link map. Once you set a relocation register, ODT uses it to relate relative addresses to absolute addresses. For more information on the relocation process, see Chapter 11.

ODT evaluates a relocatable expression as a 16-bit, six-digit (octal) number. You can type an expression in any one of the three forms presented in Table 18–1. In this table, the symbol n stands for an integer in the range 0 to 7 inclusive, and the symbol k stands for an octal number up to six digits long, with a maximum value of 177777. If you type more than six digits, ODT takes the last six digits typed, truncated to the low-order 16 bits. The symbol k may be preceded by a minus sign, in which case its value is the two's complement of the number typed. For example:

k (number typed)	Values
1	000001
–1	177777
400	000400
–177730	000050
1234567	034567

Table 18–1: Forms of Relocatable Expressions (r)

Form	Expression	Value of r
A	k	The value of k.
B	n,k	The value of k plus the contents of relocation register n. (If the n part of this expression is greater than 7, ODT uses only the last octal digit of n.)
C	C or C,k or n,C or C,C	Whenever you type the letter C, ODT replaces C with the contents of a special register called the constant register. (This value has the same role as the k or n that it replaces. The constant register is designated by the symbol $C and may be set to any value, as indicated below.)

Section 18.3.13 describes the relocation register commands in greater detail.

18.3 Commands and Functions

When ODT starts it indicates its readiness to accept commands by printing an asterisk at the left margin of the terminal. You can issue most of the ODT commands in response to the asterisk. You can examine a word and change it; you can run the object program in its entirety or in segments; you can search memory for specific words or references to them. The discussion below explains these features.

18.3.1 Printout Formats

Normally, when ODT prints addresses it attempts to print them in relative form (Form B in Table 18–1). ODT looks for the relocation register whose value is closest to, but less than or equal to, the address to be printed. It then represents the address relative to the contents of the relocation register. However, if no relocation register fits the requirement, the address prints in absolute form. Since the relocation registers are initialized to –1 (the highest number), the addresses initially print in absolute form. If you change the contents of any relocation register, it can then, depending on the command, qualify for relative form.

For example, suppose relocation registers 1 and 2 contain 1000 and 1004 respectively, and all other relocation registers contain much higher numbers. In this case, the following sequence might occur (the slash command causes the contents of the location to be printed; the line feed command, LF, accesses the next sequential location):

```
*  1000;1R                  ;sets relocation register 1 to 1000
*  1,4;2R                   ;sets relocation register 2 to 1004
*  774/000000 (LF)          ;opens location 774
000776 /000000 <LF>         ;opens location 776
1,000000 /000000 <LF>       ;absolute location 1000
1,000002 /000000 <LF>       ;absolute location 1002
2,000000 /000000            ;absolute location 1004
```

The printout format is controlled by the format register, $F. Normally this register contains 0, in which case ODT prints relative addresses whenever possible. You can open $F and change its contents to a nonzero value, however. In that case all addresses will print in absolute form (see Section 18.3.4, Accessing Internal Registers).

18.3.2 Opening, Changing, and Closing Locations

An open location is one whose contents ODT prints for examination, making those contents available for change. In a closed location, the contents are no longer available for change. Several commands are used for opening and closing locations.

Any command (except for the slash and backslash commands) that opens a location when another location is already open causes the currently open location to be closed. You can change the contents of an open location by typing the new contents followed by a single-character command that requires no argument (that is, LF, ^, RET, ←, @, >, <).

18.3.2.1 Slash (/) — One way to open a location is to type its address followed by a slash. For example:

```
*1000/012746
```

This command opens location 1000 for examination and makes it ready to be changed.

If you do not want to change the contents of an open location, press the RETURN key to close the location. ODT prints an asterisk and waits for another command. However, to change the word, simply type the new contents before giving a command to close the location. For example:

```
*1000/012746 012345 RET
*
```

This command inserts the new value, 012345, in location 1000 and closes the location. ODT prints another asterisk, indicating its readiness to accept another command.

Used alone, the slash reopens the last location opened. For example:

```
*1000/012345 2340 RET
*/002340
```

This command opens location 1000, changes its address to 002340, and then closes the location. ODT prints an asterisk, indicating its readiness to accept another command. The / character reopens the last location opened and verifies its value.

Note again that opening a location while another is open automatically closes the currently open location before opening the new location.

Also note that if you specify an odd numbered address with a slash, ODT opens the location as a byte, and subsequently behaves as if you had typed a backslash (see Section 18.3.2.2).

18.3.2.2 Backslash (\) — ODT operates on bytes, as well as on words. Typing the address of the byte followed by a backslash character opens the byte. This causes ODT to print the byte value at the specified address, to interpret the value as ASCII code, and to print the corresponding character (if possible) on the terminal. (ODT prints a ? when it cannot interpret the ASCII value as a printable character.)

```
*1001\101 =A
```

A backslash typed alone reopens the last open byte. If a word was previously open, the backslash reopens its even byte:

```
*1002/000004 \004 =?
```

18.3.2.3 LINE FEED Key (LF) — If you type the LINE FEED key when a location is open, ODT closes the open location and opens the next sequential location:

```
*1000/002340 (LF)
001002 /012740
```

In this example, the LINE FEED key caused ODT to print the address of the next location along with its contents and to wait for further instructions. After the above operation, location 1000 is closed and 1002 is open. You may modify the open location by typing the new contents.

If a byte location was open, typing a line feed opens the next byte location.

18.3.2.4 Circumflex or Up-Arrow (^) — If you type the circumflex (or up-arrow) when a location is open, ODT closes the open location and opens the previous location. To continue from the example above:

```
*001002/012740 ^
001000 /002340
```

This command closes location 1002 and opens location 1000. You may modify the open location by typing the new contents.

If the opened location was a byte, then the circumflex opens the previous byte.

18.3.2.5 Underline or Back-Arrow (_ or ←) — If you type the underline (or back-arrow) to an open word, ODT interprets the contents of the currently open word as an address indexed by the program counter (PC) and opens the addressed location:

```
*1006/000006 _
001016 /000405
```

Notice in this example that the open location, 1006, is indexed by the PC as if it were the operand of an instruction with addressing mode 67 (PC relative mode).

You can make a modification to the opened location before you type a line feed, circumflex, or underline. Also, the new contents of the location will be used for address calculations using the underline command. For example:

```
*100/000222 4(LF)          ;modifies to 4 and open next location
000102 /000111 6^          ;modifies to 6 and open previous location
000100 /000004 200_        ;changes to 200 and open location indexed
000302 /123456             ;by PC
```

18.3.2.6 Open the Addressed Location (@) – You can use the at (@) symbol to optionally modify a location, close it, and then use its contents as the address of the location to open next. For example:

```
*1006/001044 @             ;opens location 1044 next
001044 /000500

*1006/001044 2100@         ;modifies to 2100 and opens location
002100 /000167             ;2100
```

18.3.2.7 Relative Branch Offset (>) – The right-angle bracket (>) optionally modifies a location, closes it, and then uses its low-order byte as a relative branch offset to the next word to be opened. For example:

```
*1032/000407  301>         ;modifies to 301 and interprets as a
000636 /000010             ;relative branch
```

Note that 301 is a negative offset (–77). ODT doubles the offset before it adds it to the PC; therefore, 1034 + (–176) = 636.

18.3.2.8 Return to Previous Sequence (<) – The left-angle bracket (<) lets you optionally modify a location, close it, and then open the next location of the previous sequence that was interrupted by an underline, @, or right-angle bracket command. Note that underline, @, or right-angle bracket causes a sequence change to the open word. If a sequence change has not occurred, the left-angle bracket simply opens the next location as a LINE FEED does. This command operates on both words and bytes.

```
*1032/000407  301>         ;> causes a sequence change
000636 /000010 <           ;returns to original sequence
001034 /001040 @           ;@ causes a sequence change
001040 /000405 \005 = <    ;< now operates on byte
001035 \002 =? <           ;< acts like <LF>
001036 \004 =?
```

18.3.3 Accessing General Registers 0–7

Open the program's general registers 0–7 with a command in the following format:

$n/

The symbol, n, is an integer in the range 0–7 that represents the desired register. When you open these registers, you can examine them or change their contents by typing in new data, as with any addressable location. For example:

```
*$0/000033 LF          ;examines register 0 then closes it
*

*$4/000474  464RET     ;opens register 4, changes its contents
*                      ;to 000464, then closes the register
```

The example above can be verified by typing a slash in response to ODT's asterisk:

```
* 000464
```

You can use the LINE FEED, circumflex, or @ command when a register is open.

18.3.4 Accessing Internal Registers

The program's status register contains the condition codes of the most recent operational results and the interrupt priority level of the object program. Open it by typing $S. For example:

```
*$S/000311
```

$S represents the address of the status register. In response to $S in the example above, ODT prints the 16-bit word, of which only the low-order eight bits are meaningful. Bits 0–3 indicate whether a carry, overflow, zero, or negative (in that order) has resulted, and bits 5–7 indicate the interrupt priority level (in the range 0–7) of the object program. (Refer to the *PDP–11 Processor Handbook* for the Status Register format.)

You can also use the $ to open certain other internal locations listed in Table 18–2.

Table 18–2: Internal Registers

Register	Section	Contents
$B	18.3.6	First word of the breakpoint table
$M	18.3.9	Mask location for specifying which bits are to be examined during a bit pattern search
$P	18.3.15	Defines the operating priority of ODT
$S	18.3.4	Condition codes (bits 0–3) and interrupt priority level (bits 5–7)
$C	18.3.10	Constant register
$R	18.3.13	Relocation register 0, the base of the Relocation Register table
$F	18.3.1	Format register

18.3.5 Radix–50 Mode (X)

Many PDP–11 system programs employ the Radix–50 mode of packing certain ASCII characters three to a word. You can use Radix–50 mode by specifying the MACRO .RAD50 directive. ODT provides a method for examining and changing memory words packed in this way with the X command.

When you open a word and type the X command, ODT converts the contents of the opened word to its three-character Radix–50 equivalent and prints these characters on the terminal. You can then type one of the responses from Table 18–3.

Table 18–3: Radix–50 Terminators

Response	Effect
RETURN key (RET)	Closes the currently open location.
LINE FEED key (LF)	Closes the currently open location and opens the next one in sequence.
Circumflex (^)	Closes the currently open location and opens the previous one in sequence.
Any three characters whose octal code is 040 (space) or greater	Converts the three characters into packed Radix–50 format. Valid Radix–50 characters for this response are: $ Space 0 through 9 A through Z

If you type any other characters, the resulting binary number is unspecified (that is, no error message prints and the result is unpredictable). You must type exactly three characters before ODT resumes its normal mode of operation. After you type the third character, the resulting binary number is available to be stored in the opened location. Do this by closing the location in any one of the ways listed in Table 18–3. For example:

```
*1000/042431 X=KBI CBA (RET)
*1000/011421 X=CBA
```

NOTE

After ODT converts the three characters to binary, the binary number can be interpreted in one of many different ways, depending on the command that follows. For example:

```
*1234/063337 X=PRO XIT/013704
```

Since the Radix–50 equivalent of XIT is 113574, the final slash in the example will cause ODT to open location 113574 if it is a valid address.

18.3.6 Breakpoints

The breakpoint feature helps you monitor the progress of program execution. You can set a breakpoint at any instruction that is not referenced by the program for data. When a breakpoint is set, ODT replaces the contents of the breakpoint location with a BPT trap instruction so that program execution is suspended when a breakpoint is encountered. Then the original contents of the breakpoint location are restored, and ODT regains control.

With ODT you can set up to eight breakpoints, numbered 0 through 7, at any one time. Set a breakpoint by typing the address of the desired location of the breakpoint followed by ;B. Thus, r;B sets the next available breakpoint at location r. (If all eight breakpoints have been set, ODT ignores the r;B command.) You may set or change specific breakpoints by the r;nB command, where n is the number of the breakpoint. For example:

```
*1020;B      ;sets breakpoint 0
*1030;B      ;sets breakpoint 1
*1040;B      ;sets breakpoint 2
*1032;1B     ;resets breakpoint 1*
*
```

The ;B command removes all breakpoints. Use the ;nB command to remove only one of the breakpoints, where n is the number that identifies the breakpoint. For example:

```
*;2B         ;removes breakpoint 2
*
```

ODT keeps a table of breakpoints that you can access. The $B/ command opens the location containing the address of breakpoint 0. The next seven locations contain the addresses of the other breakpoints in order. You can sequentially open them by using the LINE FEED key. For example:

```
*$B/001020 LF
001136 /001032 LF
001140 /007070 LF
001142 /007070 LF
001144 /007070 LF
001146 /001046 LF
001150 /001066 LF
001152 /007070
```

In this example, breakpoint 0 is set to 1020, breakpoint 1 is set to 1032, breakpoint 5 is set to 1046, and breakpoint 6 is set to 1066. The other breakpoints are not set.

Note that a repeat count in a proceed command (;P) refers only to the breakpoint that ODT most recently encountered. Execution of other breakpoints is determined by their own repeat counts. See Section 18.3.7, below.

18.3.7 Running the Program (r;G and r;P)

ODT controls program execution. There are two commands for running the program: r;G and r;P. The r;G command starts execution (go) and r;P continues (proceed) execution after halting at a breakpoint. For example:

```
*1000;G
```

This command starts execution at location 1000. The program runs until it encounters a breakpoint or until it completes. If it gets caught in an infinite loop, it must be either restarted or reentered as explained in Section 18.1.

On execution of either the r;G or r;P command, the general registers 0–6 are set to the values in the locations specified as $0–$6. The processor status register is set to the value in the location specified as $S.

When ODT encounters a breakpoint, execution stops and ODT prints Bn; (where n is the breakpoint number), followed by the address of the breakpoint. You can then examine locations for expected data. For example:

```
*1010;3B        ;sets breakpoint 3 at location 1010
*1000;G         ;starts execution at location 1000
B3;001010       ;stops execution at location 1010
*
```

To continue program execution from the breakpoint, type ;P in response to ODT's last prompt (*).

When you set a breakpoint in a loop, you can allow the program to execute a specified number of times through the loop before ODT recognizes the breakpoint. Set a proceed count by using the r;P command. This command specifies the number of times the breakpoint is to be encountered before ODT suspends program execution (on the kth encounter). The count k refers only to the numbered breakpoint that most recently occurred. You can specify a different proceed count for the breakpoint when it is encountered. Thus:

```
B3;001010       ;halts execution at breakpoint 3
*1026;3B        ;resets breakpoint 3 at location 1026
*4;P            ;sets proceed count to 4 and
B3;001026       ;continues execution; the program loops
*               ;through the breakpoint three times and halts on
                ;the fourth occurrence of the breakpoint
```

Following the table of breakpoints (as explained in Section 18.3.6) is a table of proceed command repeat counts for each breakpoint. You can inspect these repeat counts by typing $B/ and nine line feeds. The repeat count for breakpoint 0 prints (the first seven line feeds cause the table of breakpoints to be printed; the eighth types the single-instruction mode, explained in the next section, and the ninth line feed begins the table of proceed command repeat counts). The repeat counts for breakpoints 1 through 7 and the repeat

count for the single-instruction trap follow in sequence. ODT initializes a proceed count to 0 before you assign it a value. After the command has been executed, it is set to –1. Opening any one of these provides an alternative way of changing the count. Once the location is open, you can modify its contents in the usual manner by typing the new contents followed by the RETURN key. For example:

```
.
.
.
nnnnnn /001036 LF          ;address of breakpoint 7
nnnnnn /006630 LF          ;single instruction address
nnnnnn /000000 15 LF       ;count for breakpoint 0; changes to 15
nnnnnn /000000 LF          ;count for breakpoint 1
.
.
.
nnnnnn /000000 LF          ;count for breakpoint 7
nnnnnn /nnnnnn             ;repeat count for single instruction
                           ;mode.
```

Both the address indicated as the single-instruction address and the repeat count for single-instruction mode are explained in the following section.

18.3.8 Single-Instruction Mode

With this mode, you specify the number of instructions to be executed before ODT suspends the program run. The proceed command, instead of specifying a repeat count for a breakpoint encounter, specifies the number of succeeding instructions to be executed. Note that breakpoints are disabled in single-instruction mode. Table 18–4 lists the single-instruction mode commands.

Table 18–4: Single-Instruction Mode Commands

Command	Function
;nS	Enables single-instruction mode (n can be any digit and serves only to distinguish this form from the form ;S, which disables single-instruction mode). Breakpoints are disabled.
n;P	Proceeds with program run for next n instructions before reentering ODT. (If n is missing, it is assumed to be 1.) Trapping instructions and associated handlers can affect the proceed repeat count (see Section 18.4.2).
;S	Disables single-instruction mode.

When the repeat count for single-instruction mode is exhausted and the program suspends execution, ODT prints:

```
B8;nnnnnn
```

where nnnnnn is the address of the next instruction to be executed. The $B breakpoint table contains this address following that of breakpoint 7. However, unlike the table entries for breakpoints 0–7, direct modification has no effect.

Similarly, following the repeat count for breakpoint 7 is the repeat count for single-instruction mode. You can modify this table entry directly. This is an alternative way of setting the single-instruction mode repeat count. In such a case, ;P implies the argument set in the $B repeat count table rather than an assumed 1.

18.3.9 Searches

With ODT you can search any specific portion of memory for bit patterns or references to a particular location.

18.3.9.1 Word Search (r;W) – Before initiating a word search, you must specify the mask and search limits. The location represented by $M specifies the mask of the search. $M/ opens the mask register. The next two sequential locations (opened by LINE FEEDs) initially contain the lower and upper limits of the search. ODT examines in the search all bits set to 1 in the mask and ignores other bits.

You must then give the search object and the initiating command, using the r;W command, where r is the search object. When ODT finds a match (that is, each bit set to 1 in the search object is set to 1 in the word ODT searches over the mask range), the matching word prints. For example:

```
*$M/000000 177400 (LF)          ;tests high-order eight bits
r,nnnnnn—/000000  1000 (LF)     ;sets low address limit
r,nnnnnn—/000000  1040 (RET)    ;sets high address limit
*400;W                          ;initiates word search
001010 /000770
001034 /000404
*
```

In the above example, nnnnnn is an address internal to ODT; this location varies and is meaningful only for reference purposes. In the first line above, the slash was used to open $M, which now contains 177400; the LINE FEEDs open the next two sequential locations, which now contain the upper and lower limits of the search.

In the search process, ODT performs an exclusive OR (XOR) with the word currently being examined and the search object; the result is ANDed to the mask. If this result is 0, a match has been found and ODT reports it on the terminal. Note that if the mask is 0, all locations within the limits print. This provides a convenient method for dumping all memory locations within given limits using ODT.

Typing CTRL/U during a search printout terminates the search.

18.3.9.2 Effective Address Search (r;E) – ODT provides a search for words that reference a specific location. Open the mask register only to gain access to the low- and high-limit registers. After specifying the search limits (as explained for the word search), type the command r;E (where r is the effective address) to initiate the search.

Words that are an absolute address (argument r itself), a relative address offset, or a relative branch to the effective address print after their addresses. For example:

```
*$M/177400 (LF)                ;opens mask register only to gain
r,nnnnnn /001000 1010 (LF)     ;access to search limits
r,nnnnnn /001040 1060 (RET)
*1034;E                        ;initiates search
001016 /001006                 ;relative branch
001054 /002767                 ;relative branch
*1020;E                        ;initiates a new search
001022 /177774                 ;relative address offset
001030 /001020                 ;absolute address
```

Pay particular attention to the reported effective address references. A word can have the specified bit pattern of an effective address without actually being used as one. ODT reports all possible references whether they are actually used or not.

Typing CTRL/U during a search printout terminates the search.

18.3.10 Constant Register (r;C)

It is often desirable to convert a relocatable address into its value after relocation, or to convert a number into its two's complement and then to store the converted value into one or more places in a program. Use the constant register to perform this and other useful functions.

Typing r;C evaluates the relocatable expression to its six-digit octal value, prints the value on the terminal, and stores it in the constant register. Invoke the contents of the constant register in subsequent relocatable expressions by typing the letter C. Examples follow:

```
*-4432;C=173346                ;places the two's complement of 4432 in the
                               ;constant register

*6632/062701 C (RET)           ;stores the contents of the constant
                               ;register in location 6632

*1000;1R                       ;sets relocation register 1 to 1000

*1,4272;C=005272               ;reprints relative location 4272 as an
                               ;absolute location and stores it in the
                               ;constant register
```

18.3.11 Memory Block Initialization (;F and ;I)

Use the constant register with the commands ;F and ;I to set a block of memory to a specific value. While the most common value required is 0, other possibilities are +1, –1, ASCII space, etc.

When you type the command ;F, ODT stores the contents of the constant register in successive memory words, starting at the memory word address you specify in the lower search limit and ending with the address you specify in the upper search limit.

Typing the command ;I stores the low-order eight bits in the constant register in successive bytes of memory, starting at the byte address you specify in the lower search limit and ending with the byte address you specify in the upper search limit.

For example, assume relocation register 1 contains 7000, 2 contains 10000, and 3 contains 15000. The following sequence sets word locations 7000–7776 to 0, and byte locations 10000–14777 to ASCII spaces:

```
                                  ;opens the mask register to gain
*$M/ 000000 (LF)                  ;access to search limits
r,nnnnnn  /000000 1,0 (LF)        ;sets the lower limit to 7000
r,nnnnnn  /000000 2,-2 (RET)      ;sets the upper limit to 7776
*0;C=000000                       ;sets the constant register to zero
*;F                               ;sets locations 7000–7776 to zero

*$M/ 000000 (LF)
r,nnnnnn  /007000 2,0 (LF)        ;sets the lower limit to 10000
r,nnnnnn  /007776 3,-1 (RET)      ;sets the upper limit to 14777
*40;C=000040                      ;sets the constant register to 40
                                  ;(space)
*;I                               ;sets the byte locations
                                  ;10000–14777
*                                 ;to the value in the low-order
                                  ;eight bits of the constant
                                  ;register
```

18.3.12 Calculating Offsets (r;O)

Relative addressing and branching involve the use of an offset. An offset is the number of words or bytes forward or backward from the current location to the effective address. During the debugging session it is sometimes necessary to change a relative address or branch reference by replacing one instruction offset with another. ODT calculates the offsets in response to the r;O command.

The command r;O causes ODT to print the 16-bit and 8-bit offsets from the currently open location to address r. For example:

```
*346/ 000034 414;O  000044 022 22  (RET)
*/ 000022
```

This command opens location 346, calculates and prints the offsets from location 346 to location 414, changes the contents of location 346 to 22 (the 8-bit offset), and verifies the contents of location 346.

The 8-bit offset prints only if it is in the range –128(decimal) to 127(decimal) and the 16-bit offset is even, as was the case above. In the next example, the offset of a relative branch is calculated and modified so that it branches to itself.

```
*1034/103421 1034;O  177776 377 \021 =?  377(RET)
*/103777
```

Note that the modified low-order byte 377 must be combined with the unmodified high-order byte.

18.3.13 Relocation Register Commands

The use of the relocation registers is described briefly in Section 17.2. At the beginning of a debugging session it is desirable to preset the registers to the relocation biases of those relocatable modules that will be receiving the most attention. Do this by typing the relocation bias, followed by a semicolon and the specification of relocation registers, using the following syntax:

r;nR

The symbol r may be any relocatable expression, and n is an integer in the range 0–7. If you omit n, it is assumed to be 0. For example:

```
*1000;5R          ;puts 1000 into relocation register 5
*5,100;5R         ;adds 100 to the contents
*                 ;of relocation register 5
```

Once a relocation register is defined, you can use it to reference relocatable values. For example:

```
*2000;1R          ;puts 2000 into relocation register 1
*1,2176/002466    ;examines the contents of location 4176
*1,3712;0B        ;sets a breakpoint at location 5712
```

Sometimes programs may be relocated to an address below the one at which they were assembled. This could occur with PIC code (position-independent code), which is moved without using the linker. In this case, the appropriate relocation bias would be the two's complement of the actual downward displacement. One method for easily evaluating the bias and putting it in the relocation register is illustrated in the following example.

Assume a program was assembled at location 5000 and was moved to location 1000. Then the following sequence enters the two's complement of 4000 in relocation register 1.

```
*1000;1R
*1,-5000;1R
*
```

Relocation registers are initialized to –1 so that unwanted relocation registers never enter into the selection process when ODT searches for the most appropriate register.

To set a relocation register to –1, type ;nR. To set all relocation registers to –1, type ;R.

ODT maintains a table of relocation registers, beginning at the address specified by $R. Opening $R ($R/) opens relocation register 0. Successively typing a LINE FEED opens the other relocation registers in sequence. When a relocation register is opened in this way, you can modify it as you would any other memory location.

18.3.14 The Relocation Calculators, n! and nR

When a location has been opened, it is often desirable to relate the relocated address and the contents of the location back to their relocatable values. To calculate the relocatable address of the opened location relative to a particular relocation bias, use the following syntax:

n!

The symbol n specifies the relocation register. This calculator works with opened bytes and words. If you omit n, the relocation register whose contents are closest to, but less than or equal to, the opened location is selected automatically by ODT. In the following example, assume that these conditions are fulfilled by relocation register 3, which contains 2000. Use the following command to find the most likely module that a given opened byte is in:

```
* 2500\011 =  !=3,000500
```

To calculate the difference between the contents of the opened location and a relocation register, use the following syntax:

nR

The symbol n represents the relocation register. If you omit n, ODT selects the relocation register whose contents are closest to, but less than or equal to, the contents of the opened location. For example, assume the relocation bias stored in relocation register 1 is 7000:

```
* 1,500/011032  1R=1,002032
```

The value 2032 is the content of 1,500, relative to the base 7000. The next example shows the use of both relocation calculators.

If relocation register 1 contains 1000, and relocation register 2 contains 2000, use the following command to calculate the relocatable addresses of location 3000 and its contents, relative to 1000 and 2000:

```
* 3000/006410  1!=1,002000  2!=2,001000  1R=1,005410  2R=2,00410
```

18.3.15 ODT Priority Level ($P)

$P represents a location in ODT that contains the interrupt (or processor) priority level at which ODT operates. If $P contains the value 377, ODT operates at the priority level of the processor at the time ODT is entered. Otherwise $P may contain a value between 0 and 7 corresponding to the fixed priority at which ODT operates.

To set ODT to the desired priority level, open $P. ODT prints the present contents, which you can then change:

```
* $P/000006  4 (RET)        ;lowers the priority to allow interrupts
*                           ;from the terminal
```

If you do not change $P, its value is seven.

You must set ODT's priority to 0 if you are using ODT in a foreground/background environment while another job is running.

ODT may not always service breakpoints that are set in routines that run at different priority levels. For example, a program running at a low priority can use a device service routine that operates at a higher priority level. If you set $P low, ODT waits for terminal input at a low priority. If an interrupt occurs from a high-priority routine, the breakpoints in the high-priority routine will not be recognized because they were removed when the earlier breakpoint occurred. Thus, interrupts that are set at a priority higher than the one at which ODT is running will be serviced, but any breakpoints will not be recognized. To avoid this problem, set breakpoints at one priority level at a time. That is, set breakpoints within an interrupt service routine, but not at mainline code level. For a more complete discussion of how the PDP–11 handles priority and interrupts, refer to the processor handbook for your particular machine. ODT disables all breakpoints in the program whenever it gains control. Breakpoints are enabled when ;P and ;G commands are executed. For example:

```
*$P/00007 5
*1000;B
*2000;B
*1000;G
B0:001000
*                          ;an interrupt occurs and is serviced
```

If a higher-level interrupt occurs while ODT is waiting for input, the interrupt is serviced, and no breakpoints are recognized.

18.3.16 ASCII Input and Output (r;nA)

Inspect and change ASCII text by using a command of this syntax:

r;nA

The symbol r represents a relocatable expression, and n is a character count. If you omit n, it is assumed to be 1. ODT prints n characters starting at location r and followed by a carriage return/line feed combination. Table 18–5 lists responses and their effect.

Table 18–5: ASCII Terminators

Response	Effect
RETURN Key (<RET>)	ODT outputs a carriage return/ line feed combination followed by an asterisk, and waits for another command.
LINE FEED Key (<LF>)	ODT opens the byte following the last byte that was output.
Up to n characters of text	ODT inserts the text into memory, starting at location r. If you type exactly n characters, ODT responds with <CR><LF> address <CRXLF>*. If you type fewer than n characters, terminate that string with CTRL/U. ODT responds with ?^U <CR><LF> address <CR><LF>.

18.4 Programming Considerations

Information in this section is not necessary for normal use of ODT. However, it does provide a better understanding of how ODT performs some of its functions. In certain difficult debugging situations, this understanding is necessary.

18.4.1 Using ODT with Foreground/Background Jobs

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the SJ monitor, you can link ODT with the program as described in the first example in Section 18.1. To debug a program in the foreground area, DIGITAL recommends that you run ODT in the background while the program to be debugged is in the foreground. The sequence of commands to do this is:

```
.FRUN PROG/P                ;loads the foreground program
LOADED AT nnnnnn            ;the first address of the job prints
.RUN ODT                    ;runs ODT in the background
ODT V01.01                  ;and sets a relocation register
*nnnnnn;0R                  ;to the start of the job

*$F/000000 0                ;clears the format register to enable
*0,nnnnnn;0B                ;proper address printing
                            ;sets a breakpoint

*0;G                        ;starts the keyboard monitor again

.RESUME                     ;starts the foreground job
```

The copy of ODT used must be linked low enough so that it fits in memory along with the foreground job.

NOTE

> Since ODT uses its own terminal handler, it cannot be used with the display hardware. If GT ON is in effect, ODT ignores it and directs its input and output only to the console terminal.

If you use ODT in a foreground/background environment while another job is running, set ODT's priority bit to 0 as follows:

```
*$P/000007 0 (RET)
```

This puts ODT into the wait state at level 0, not at level 7. If you leave ODT's priority at 7, all interrupts (including clock) are locked out while ODT is waiting for terminal input.

18.4.2 Functional Organization

The internal organization of ODT is almost totally modularized into independent subroutines. The internal structure consists of three major functions: command decoding, command execution, and utility routines.

The command decoder interprets the individual commands, checks for command errors, saves input parameters for use in command execution, and sends control to the appropriate command execution routine.

The command execution routines take parameters saved by the command decoder and use the utility routines to execute the specified command. Command execution routines either return to the command decoder or transfer control to your program.

The utility routines are common routines such as SAVE-RESTORE and I/O. They are used by both the command decoder and the command executers.

18.4.3 Breakpoints

The function of a breakpoint is to give control to ODT whenever a program tries to execute the instruction at the selected address.

When a breakpoint is executed, ODT removes all the breakpoint instructions from the code so that you can examine and alter the locations. ODT then types a message on the terminal in the form Bn;r, where r is the breakpoint address and n is the breakpoint number. ODT restores the breakpoints when execution resumes.

There is a major restriction in the use of breakpoints: the program must not reference the word where a breakpoint was set since ODT altered the word. You should also avoid setting a breakpoint at the location of any instruction that clears the T-bit. For example:

```
MOV #240,177776          ;SET PRIORITY TO LEVEL 5
```

NOTE

> Instructions that cause traps or returns from them (for example, EMT, RTI) are likely to clear the T-bit, because a new word from the trap vector or the stack is loaded into the status register.

A breakpoint occurs when a trace trap instruction (placed in your program by ODT) is executed. When a breakpoint occurs, ODT operates according to the following algorithm:

1. Sets processor priority to seven (automatically set by trap instruction).
2. Saves registers and sets up stack.
3. If internal T-bit trap flag is set, goes to step 13.
4. Removes breakpoints.
5. Resets processor priority to ODT's priority or user's priority.
6. Makes sure a breakpoint or single-instruction mode caused the interrupt.
7. If the breakpoint did not cause the interrupt, goes to step 15.

8. Decrements repeat count.
9. Goes to step 18 if nonzero; otherwise resets count to one.
10. Saves terminal status.
11. Types message about the breakpoint or single-instruction mode interrupt.
12. Goes to command decoder.
13. Clears T-bit in stack and internal T-bit flag.
14. Jumps to the go processor.
15. Saves terminal status.
16. Types BE (bad entry), followed by the address.
17. Clears the T-bit, if set, in the user status and proceeds to the command decoder.
18. Goes to the proceed processor, bypassing the TT restore routine.

Note that steps 1–5 inclusive take approximately 100 microseconds. Interrupts are not permitted at this time, because ODT is running at priority level 7.

ODT processes a proceed (;P) command according to the following algorithm:

1. Checks the proceed for validity.
2. Sets the processor priority to seven.
3. Sets the T-bit flags (internal and user status).
4. Restores the user registers, status, and program counter.
5. Returns control to the user.
6. When the T-bit trap occurs, executes steps 1, 2, 3, 13, and 14 of the breakpoint sequence, restores breakpoints, and resumes normal program execution.

When a breakpoint is placed on an IOT, EMT, TRAP, or any instruction causing a trap, ODT follows this algorithm:

1. When the breakpoint occurs as described above, enters ODT.
2. When ;P is typed, sets the T-bit and executes the IOT, EMT, TRAP, or other trapping instruction.
3. Pushes the current PC and status (with the T-bit included) on the stack.
4. Obtains the new PC and status (no T-bit set) from the respective trap vector.
5. Executes the whole trap service routine without any breakpoints.

6. When an RTI is executed, restores the saved PC and PS (including the T-bit). Executes the instruction following the trap-causing instruction. If this instruction is not another trap-causing instruction, the T-bit trap occurs; reinserts the breakpoints in the user program, or decrements the single-instruction mode repeat count. If the following instruction is a trap-causing instruction, repeats this sequence starting at step 3.

NOTE

> Exit from the trap handler must be by means of the RTI instruction. Otherwise, the T-bit is lost. ODT cannot regain control because the breakpoints have not yet been reinserted.

Note that the ;P command is invalid if a breakpoint has not occurred (ODT responds with ?). ;P is valid, however, after any trace trap entry.

The internal breakpoint status words have the following format:

1. The first eight words contain the breakpoint addresses for breakpoints 0–7. (The ninth word contains the address of the next instruction to be executed in single-instruction mode.)

2. The next eight words contain the respective repeat counts. (The following word contains the repeat count for single-instruction mode.)

You may change these words at will, either by using the breakpoint commands or by directly manipulating $B.

When program runaway occurs (that is, when the program is no longer under ODT control, perhaps executing an unexpected part of the program where you did not place a breakpoint), give control to ODT by pressing the HALT key to stop the computer and then restarting ODT (see Section 18.1). ODT prints an asterisk, indicating that it is ready to accept a command.

If the program you are debugging uses the console terminal for input or output, the program can interact with ODT to cause an error because ODT uses the console terminal as well. This interactive error does not occur when you run the program without ODT.

Note the following rules concerning the ODT break routine:

1. If the console terminal interrupt is enabled upon entry to the ODT break routine, and no output interrupt is pending when ODT is entered, ODT generates an unexpected interrupt when returning control to the program.

2. If the interrupt of the console terminal reader (the keyboard) is enabled upon entry to the ODT break routine, and the program is expecting to receive an interrupt to input a character, both the expected interrupt and the character are lost.

3. If the console terminal reader (keyboard) has just read a character into the reader data buffer when the ODT break routine is entered, the expected character in the reader data buffer is lost.

18.4.4 Searches

The word search lets you search for bit patterns in specified sections of memory. Using the $M/ command, specify a mask, a lower search limit ($M+2), and an upper search limit ($M+4). Specify the search object in the search command itself.

The word search compares selected bits (where 1s appear in the mask) in the word and search object. If all of the selected bits are equal, the unmasked word prints.

The following shows the search algorithm.

1. Fetches a word at the current address.
2. XORs (exclusive OR) the word and search object.
3. ANDs the result of step 2 with the mask.
4. If the result of step 3 is zero, types the address of the unmasked word and its contents; otherwise, proceeds to step 5.
5. Adds two to the current address. If the current address is greater than the upper limit, types * and returns to the command decoder; otherwise, goes to step 1.

Note that if the mask is 0, ODT prints every word between the limits, since a match occurs every time (that is, the result of step 3 is always 0).

In the effective address search, ODT interprets every word in the search range as an instruction that is interrogated for a possible direct relationship to the search object. The mask register is opened only to gain access to the search limit registers.

The algorithm for the effective address search is as follows ((X) denotes contents of X, and K denotes the search object):

1. Fetches a word at the current address X.
2. If (X) = K [direct reference], prints contents and goes to step 5.
3. If (X) + X + 2 = K [indexed by PC], prints contents and goes to step 5.
4. If (X) is a relative branch to K, prints contents.
5. Adds 2 to the current address. If the current address is greater than the upper limit, performs a carriage return/line feed combination and returns to the command decoder; otherwise, goes to step 1.

18.4.5 Terminal Interrupt

When entering the TT SAVE routine, ODT follows these steps:

1. Saves the LSR status register (TKS).
2. Clears interrupt enable and maintenance bits in the TKS.

3. Saves the TT status register (TPS).

4. Clears interrupt enable and maintenance bits in the TPS.

To restore the TT:

1. Wait for completion of any I/O from ODT.

2. Restore the TKS.

3. Restore the TPS.

NOTE

If the TT printer interrupt is enabled upon entry to the ODT break routine, the following can occur:

1. If no output interrupt is pending when ODT is entered, an additional interrupt always occurs when ODT returns control to the user.

2. If an output interrupt is pending upon entry, the expected interrupt occurs when the user regains control.

If the TT reader (keyboard) is busy or done, the expected character in the reader data buffer is lost.

If the TT reader (keyboard) interrupt is enabled upon entry to the ODT break routine, and a character is pending, the interrupt (as well as the character) is lost.

18.5 Error Detection

ODT detects two types of error: invalid or unrecognizable command and bad breakpoint entry. ODT does not check for the validity of an address when you command it to open a location for examination or modification. Thus in the following example, the command references nonexistent memory, thereby causing a trap through the vector at location 4.

```
177774/
?MON-F-Trap to 4 003362
```

If the program you are debugging with ODT has requested traps through location 4 with the .TRPSET EMT, the program receives control at its TRPSET address.

If something other than a valid command is typed, ODT ignores the command and prints:

```
(echoes invalid command)?
*
```

ODT then waits for another command. Therefore, to cause ODT to ignore a command that has just been typed, type any invalid character (such as 9 or RUBOUT), and the command will be treated as an error and ignored.

ODT suspends program execution whenever it encounters a breakpoint (that is, traps to its breakpoint routine). If the breakpoint routine is entered and no known breakpoint caused the entry, ODT prints:

```
BEnnnnnn
*
```

and waits for another command. BEnnnnnn denotes bad entry from location nnnnnn. A bad entry may be caused by an invalid trace trap instruction, by a T-bit set in the status register, or by a jump to some random location within ODT.

Chapter 19
Object Module Patch Program (PAT)

The RT–11 object module patch program (PAT) allows you to update code in a relocatable binary object module (.OBJ file). PAT does not permit you to examine the octal contents of an object module. PAT makes the patch to the object module by means of the procedure outlined in Figure 19–2. One advantage to using PAT is that you can add relatively large patches to an object module without performing any octal calculations. PAT accepts a file containing corrections or additional instructions and applies these corrections and additions to the original object module. You prepare correction input in source form and assemble it with the MACRO–11 assembler.

Two files form the input to PAT: the original input file, and a correction file containing the corrections and additions to that input file. The original input file consists of one or more concatenated object modules, only one of which can be corrected with a single execution of the PAT utility. The correction file consists of object code that, when linked by the linker, either replaces or appends to the original object module. Output from PAT is the updated input file.

It is always good practice to create a backup version of the file you want to patch before you use PAT to make the changes.

19.1 Calling and Using PAT

To call PAT from the system device, respond to the dot (.) printed by the keyboard monitor by typing:

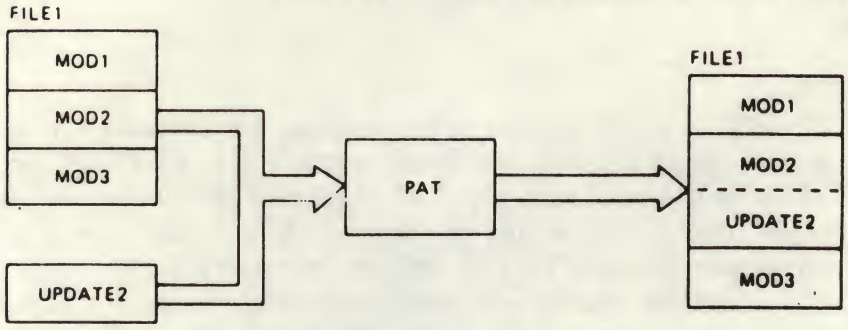
```
.R PAT (RET)
```

The Command String Interpreter (CSI) prints an asterisk at the left margin of the console terminal when it is ready to accept a command line. Chapter 1 describes the general syntax of the command line PAT accepts.

Type two CTRL/Cs to halt PAT at any time (or a single CTRL/C to halt PAT when it is waiting for console terminal input) and return control to the monitor. To restart PAT, type R PAT in response to the monitor's dot. When PAT completes an update operation it returns control to CSI level (*).

Figure 19–1 shows how you use PAT to update a file (FILE1) consisting of three object modules (MOD1, MOD2, and MOD3) by appending a correction file to MOD2. After running PAT, you use the linker to relink the updated module with the rest of the file and to produce a corrected executable program.

Figure 19–1: Updating a Module Using PAT



There are several steps you must follow when using PAT to update a file. First, use a text editor to create the correction file. Then, assemble the correction file to produce an object module. Next, submit the input file and the correction file in object module form to PAT for processing. Finally, link the updated object module, along with the object modules that make up the rest of the file, to resolve global symbols and create an executable program. Figure 19–2 shows the processing steps involved in generating an updated executable file using PAT.

19.2 PAT Command String Syntax

Specify the PAT command string in the following form:

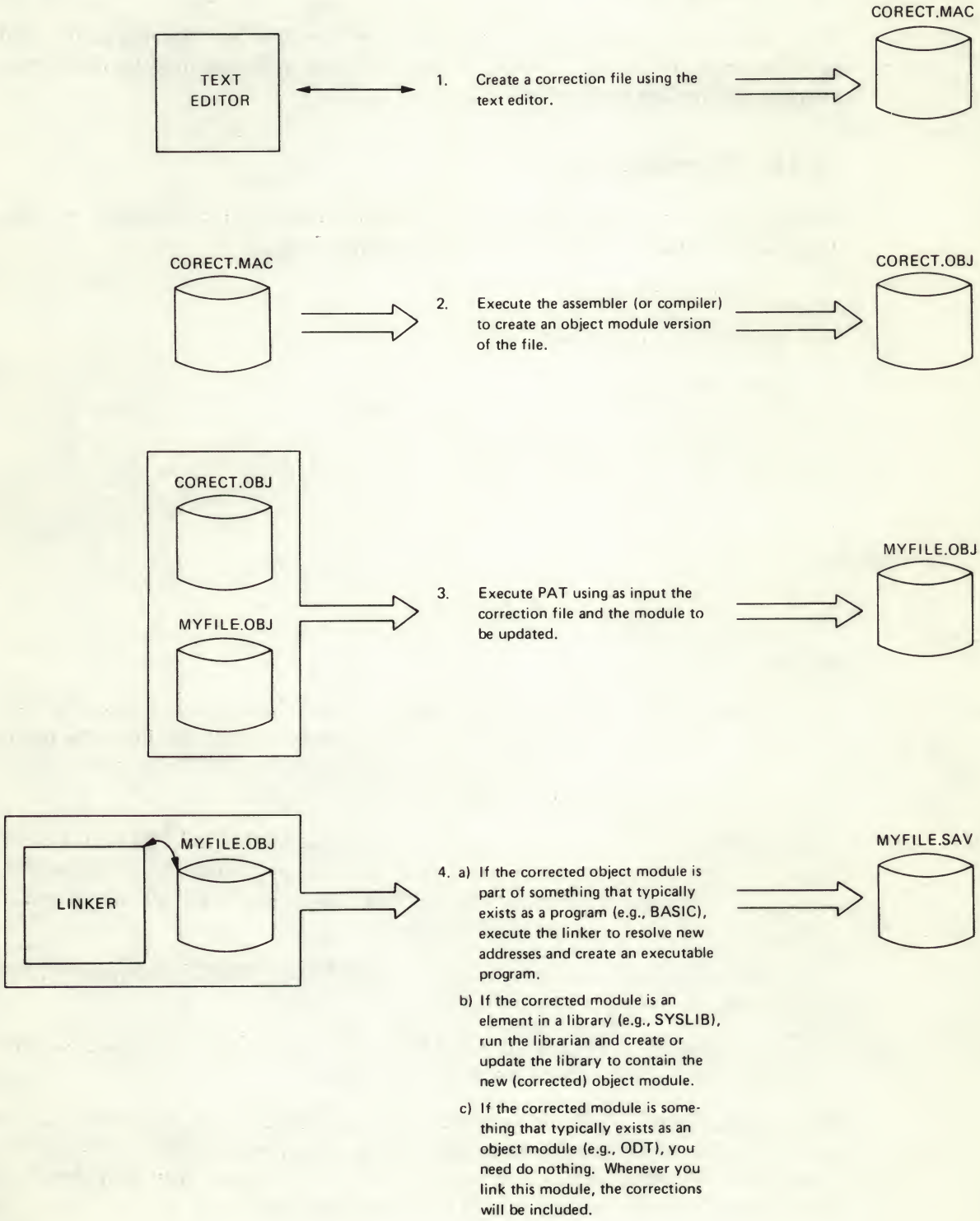
[output-filespec] = input-filespec[/C[:n]],correct-filespec[/C[:n]]

where:

output-filespec	is the file specification for the output file. If you do not specify an output file, PAT does not generate one.
input-filespec	is the file specification for the input file. This file can contain one or more concatenated object modules.
correct-filespec	is the file specification for the correction file. This file contains the updates being made to a single module in the input file.
/C	specifies the checksum option for the associated file. This directs PAT to generate an octal value for the sum of all the binary data composing the module in that file. (See Section 19.5 for more information on checksums.)

n specifies an octal value. PAT compares the checksum value it computes for a module with the octal value you specify.

Figure 19–2: Processing Steps Required to Update a Module Using PAT



19.3 How PAT Effects Updates

PAT updates a base input module by using additions and corrections you supply in a correction file. This section describes the PAT input and correction files, and gives information on how to create the correction file.

19.3.1 Input File

The input file is the file to be updated; it is the base for the output file and must be in object module format. When PAT executes, the module in the correction file applies to this file.

19.3.2 Correction File

The correction file must be in object module format and it is usually created from a MACRO–11 source file in the following format:

```
.TITLE inputname
[.IDENT updatenum]
[section name]
inputline
inputline
    *
    *
    *
```

where:

inputname	is the name of the module to be corrected by the PAT update. That is, inputname must be the same name as the name on the input file .TITLE directive for a single module in the input file.
updatenum	is any value acceptable to the MACRO–11 assembler. Generally, this value reflects the update version of the file being processed by PAT, as shown in the examples below.
section name	is the ASECT, CSECT, or PSECT included in the correction file.
inputline	are lines of input for PAT's use in correcting and updating the input file.

During execution, PAT adds any new global symbols that are defined in the correction file to the module's symbol table. Duplicate global symbols in the correction file supersede their counterparts in the input file, provided that both definitions are relocatable or both are absolute.

A duplicate PSECT or CSECT supersedes the previous PSECT or CSECT, provided:

- Both have the same relocatability attribute (ABS or REL).
- Both are defined with the same directive (.PSECT or .CSECT).

If PAT encounters duplicate PSECT names, it sets the length attribute for the PSECT to the length of the longer PSECT and appends a new PSECT to the module.

If you specify a transfer address, it supersedes that of the module you are patching.

19.4 Updating Object Modules

The following examples show the source code for an input file and a correction file to be processed by PAT and the linker. The examples show as output a single source file that, if assembled and linked, would produce a binary module equivalent to the file generated by PAT and LINK. Two techniques are described: one is for overlaying lines in a module, and the other is for appending a subroutine to a module.

19.4.1 Overlaying Lines in a Module

In the following example, PAT first appends the correction file to the input file. The linker is then executed to replace code within the input file.

The input file for this example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
        .END
```

To add the instruction ADD A,B after the JSR instruction, the following patch source file is included:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
.=.+12
        ADD     A,B
        RTS     PC
        .END
```

The patch source is assembled using MACRO–11 and the resulting object file is input to PAT along with the original object file. The following source code represents the result of PAT processing:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        RTS     PC
.=ABC
.=.+12
        ADD     A,B
        RTS     PC
        .END
```

After the linker processes these files, the load image appears, as this source code representation shows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        MOV     A,C
        JSR     PC,XYZ
        ADD     A,B
        RTS     PC
        .END
```

The linker uses the . = . + 12 in the program counter field to determine where to begin overlaying instructions in the program and, finally, overlays the RTS instruction with the patch code:

```
ADD     A,B
RTS     PC
```

19.4.2 Adding a Subroutine to a Module

In many cases, a patch requires that more than a few lines be added to patch the file. A convenient technique for adding new code is to append it to the end of the module in the form of a subroutine. This way, you can insert a JSR instruction to the subroutine at an appropriate location. The JSR directs the program to branch to the new code, execute that code, and then return to in-line processing.

The source code for the input file for the example is:

```
        .TITLE  ABC
        .IDENT  /01/
        .ENABL  GBL
ABC::
        MOV     A,B
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .END
```

Suppose you wish to add the instructions:

```
MOV     D,R0
ASL     R0
```

between

```
MOV     A,B
```

and

```
JSR     PC,XYZ
```

The correction file to accomplish this is as follows:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
        JSR     PC,PATCH
        NOP
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

PAT appends the correction file to the input file, and the linker then processes the file, generating the following output file:

```
        .TITLE  ABC
        .IDENT  /01.01/
        .ENABL  GBL
ABC::
        JSR     PC,PATCH
        NOP
        JSR     PC,XYZ
        MOV     C,R0
        RTS     PC
        .PSECT  PATCH
PATCH:
        MOV     A,B
        MOV     D,R0
        ASL     R0
        RTS     PC
        .END
```

In this example, the JSR PC,PATCH and NOP instructions overlay the three-word MOV A,B instruction. (The NOP is included because this is a case where a two-word instruction replaces a three-word instruction. NOP is required to maintain alignment.) The linker allocates additional storage for .PSECT PATCH, writes the specified code into this program section, and binds the JSR instruction to the first address in this section. Note that the MOV A,B instruction, replaced by the JSR PC,PATCH, is the first instruction the PATCH subroutine executes.

19.5 Determining and Validating the Contents of a File

Use the checksum option (/C) to determine or validate the contents of a module. The checksum option directs PAT to compute the sum of all binary data composing a file. If you specify the command in the form /C:n, /C directs PAT to compute the checksum and compare that checksum to the value you specify as n.

To determine the checksum of a file, enter the PAT command line with the /C option applied to the appropriate file (the file whose checksum you want to determine). For example, PAT responds to the command

```
=INFILE/C,INFILE.PAT
```

with the message

```
?PAT-W-Input module checksum is nnnnnn
```

PAT generates a similar message when you request the checksum for the correction file.

To validate the changes made to a file, enter the checksum option in the form /C:n. PAT compares the value it computes for the checksum with the value you specify as n. If the two values do not match, PAT enters the changes but displays a message reporting the checksum error as either:

```
?PAT-W-Input file checksum error
```

or

```
?PAT-W-Correction file checksum error
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

Chapter 20
Save Image Patch Program (SIPP)

The save image patch program (SIPP) lets you make code modifications to any RT–11 file that exists on a random-access storage volume. You use SIPP primarily for maintaining save image files. Although SIPP is designed for maintaining programs that have been created with the RT–11 Version 4 or later linker, you can use SIPP for pre–Version 4 programs that are not overlaid.

SIPP is also useful for examining locations within a file. If you do not modify any locations within a file, SIPP makes no changes. Also, you can run SIPP from an indirect command file, a BATCH stream, or from the console.

When you run SIPP, you have the option of installing your code modifications when you close the file, or you can create a command file that contains both the code modifications and the instructions necessary for SIPP to install them. You can run this command file as an indirect file whenever you wish. When SIPP patches a file, the creation date of the patched file is changed to the current system date.

Because SIPP does not install code modifications until you have finished making them, SIPP's checksum is not affected by a CTRL/U or DELETE. This feature also makes the code modification, or patching, procedure easier for you.

NOTE

DIGITAL does not recommend that you modify the following data within a save image file: locations 50, 64, and 66; the Job Status Word; the overlay handler; the overlay tables; and the window definition blocks. SIPP uses these locations for internal calculations and will automatically update them as necessary. Note, however, that if you use the /A option, SIPP does not modify any of these locations.

20.1 Calling and Using SIPP

To call SIPP, respond to the dot (.) printed by the keyboard monitor by typing:

```
.R SIPP (RET)
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, SIPP prints its current version number. If you type a CTRL/C in response to the asterisk, control returns to the monitor. If you type a CTRL/C in response to any of SIPP's prompts, SIPP prints the following confirmation message:

```
?SIPP -  Are you sure?
```

If you type Y or any string beginning with Y followed by a carriage return, SIPP aborts the patching procedure, and returns control to the monitor, without making any changes to your file. Any other response returns control to the procedure that was interrupted. You must type two consecutive CTRL/Cs at any other time, including while running from an indirect command file, to get the ?SIPP - Are you sure? message.

Enter a command string according to this general syntax:

[com-filespec =]input-filespec[/option...]

where:

com-filespec	represents the file specifications of the command file that you want SIPP to create. You can run this file as an indirect file. The default file type is .COM. If you do not specify a command file, SIPP does not create one.
input-filespec	represents the file specifications of the file you want to modify. If you do not specify a file type, SIPP assumes .SAV.
/option	is one of the options listed in Table 20–1.

If you enter only a device specification in response to the CSI asterisk, SIPP opens the first block of that volume and assumes the /A option.

20.2 SIPP Options

Table 20–1 summarizes the options that you can use in the CSI command string to SIPP.

20.3 SIPP Dialog

After you have entered the initial command string to SIPP, SIPP prints a series of prompts at the terminal. The responses you give to these prompts guide SIPP to the location in the input file or volume where you want to begin code modifications. If the input file is overlaid, the first prompt SIPP prints at the terminal is:

```
Segment?
```

Table 20–1: SIPP Options

Option	Function
/A	Prevents SIPP from automatically modifying either location 50, the window definition blocks, the overlay table, or the overlay handler. Use /A when you are patching anything other than save image files. When you use the option, SIPP modifies only those locations that you specify.
/C	Requires you to enter a checksum after you finish code modifications. If you make no modifications, SIPP ignores /C. The command file will automatically contain /C. You cannot use /C and /D together. See Section 20.5 for more details on the checksum.
/D	Use if you do not know the checksum for a particular patch and you want SIPP to create one. SIPP prints the checksum for the patch after you have finished entering all the code modifications. If you make no modifications, SIPP ignores the /D option. You cannot use /C and /D together.
/L	When you use /L, SIPP does not modify the input file after the patching session. This option is useful if you wish only to create a command file and preserve the input file.

Respond to this prompt by typing the number of the overlay segment that contains the locations you want to modify. (SIPP does not print this prompt: if the file you are modifying is not overlaid, if you are using the /A option, or if you are modifying a volume.) You can find the segment number in the program's load map. Type a carriage return, or 0 followed by a carriage return, if you want to modify the program's root segment.

SIPP prompts you for the base address within the program or overlay segment where you want to begin code modifications or examination. SIPP prints the following prompt for both overlaid and non-overlaid files. (Note that the following prompt is the second prompt for overlaid files, and the first prompt for anything else.)

```
Base?
```

If the file you are modifying is overlaid, respond to the last prompt by entering the base address specified on the load map for the segment you want to modify. If the file is not overlaid, enter the load address of the program section you wish to modify or examine.

After you have entered the base address, SIPP prompts you for the offset as follows:

```
Offset?
```

Respond to the offset prompt by typing the offset from the current base where you want to begin modifying or examining your program.

If the offset you specify is an even number, SIPP opens the corresponding location as a word. If the offset is odd, SIPP opens the location as a byte.

Section 20.4 describes how you can alternate between words and bytes as you proceed to modify or examine the file.

After you have responded to Offset?, SIPP prints the following header:

```
Segment        Base      Offset      Old   New?
```

If the file is not overlaid, SIPP does not print the segment column. Below the header, SIPP prints the segment, base, and offset you have specified by responding to the dialog prompts.

A sample dialog format follows. In this example, SIPP is to begin code modifications in overlay segment 2 of program PROG.SAV.

```
.R SIPP(RET)
*PROG=PROG(RET)
Segment? 2(RET)
Base?    20000(RET)
Offset?  100(RET)
Segment      Base      Offset      Old   New?
000002       20000     20100     103425
```

Under the column marked Old, SIPP prints the contents of the currently open location. Under the column designated New?, you can enter either a new value for the current location and/or a command. Section 20.4 gives more details on opening and modifying locations. Table 20–2 summarizes the commands you can enter.

NOTE

SIPP does not make changes to a file as you type them. Instead, SIPP stores the changes in a buffer, allowing you to abort a partially completed patch operation without leaving behind a partially patched file. When you finish a patching operation by typing CTRL/Y or multiple CTRL/Zs (see Table 20–2), SIPP makes all the changes in one pass.

20.4 SIPP Commands

Table 20–2 summarizes the commands you can enter during the code modification procedure and lists the sections in which you can find more details on each command. You can follow command with either a line feed or a carriage return.

Table 20–2: SIPP Commands

Command	Section	Function
<RET> or <LF>	20.4.1	Closes the current location without modifying it, and opens and displays the next location.
n<RET>	20.4.1	Enters the value represented by n in the current location, closes it, and opens the next location.

(Continued on next page)

Table 20–2: SIPP Commands (Cont.)

Command	Section	Function
^<RET>	20.4.2	Closes the current location without modifying it, and opens the previous location.
n^<RET>	20.4.2	Enters the value represented by n in the current location, closes it, and opens the previous location.
\<RET>	20.4.3	Reopens the current location as a byte (starting with the low, or even, byte for that word). From this point, SIPP will continue opening byte locations and accepting byte values. Do not use this command when in Radix–50 or ASCII mode.
/<RET>	20.4.3	Reopens the current location as a word. SIPP displays the contents of the currently open word location. All further displays and input will be word values. Do not use this command when in Radix–50 or ASCII mode.
;O<RET>	20.4.4	Reopens the current location as an octal word value. This is the default mode. Use ;O to return to octal after having been in Radix–50 or ASCII mode. All further displays and input are octal values.
;A<RET>	20.4.5	Displays the byte of the current location as an ASCII value. All further displays are in ASCII, and SIPP advances in byte mode.
;Ax<RET>	20.4.5	Inserts an ASCII character represented by x in the byte of the current location, closes that byte, and opens and displays the next location. Use this command for inserting only one ASCII character at a time. You can also use this command to search for an ASCII value (see Section 20.4.7).
;R<RET>	20.4.6	Displays the current location as a word of up to three Radix–50 characters. All further displays are in Radix–50.
;Ryyy<RET>	20.4.6	Inserts up to three Radix–50 characters represented by yyy into the current location. SIPP then closes the current location, and opens and displays the next location. Use this command for inserting up to three Radix–50 characters. You can also use this command to search for a Radix–50 value (see Section 20.4.7).
;S<RET>	20.4.7	Searches for a value within the file. When you type this command, SIPP prompts you for a value for which it is to search. SIPP also prompts you for the boundaries within which you want it to conduct the search.
;V<RET>	20.4.8	Prints all the modifications you have made in the current patching session. You can use this command at any time, except in response to Checksum?.
CTRL/Z<RET>	20.4.9	Backs up to the previous prompt: Offset?, Base?, or Segment?. This command allows you insert code modifications in more than one area of the file during the same patching session.
CTRL/Y<RET>	20.4.10	Completes the current patching session, installs the patch, creates the command file (if requested), and prompts you with an asterisk for another file specification.

20.4.1 Opening and Modifying Locations Within a File

As stated earlier, after you guide SIPP to the location in the file where you want to begin making code modifications, SIPP prints out a header under which it lists the address and contents of the location specified, called the current location. Under the last column in the header, New?, you can enter a value that replaces the contents of the current location. After you enter the new value, type a carriage return to advance to the next location.

In the following example, the value 240 is inserted in the current location. Next, a carriage return advances SIPP to the next 16-bit location.

```
Base        Offset        Old    New?
 001000      001200    004767    240RET
 001000      001202    003106
```

If you do not want to modify the current location, simply type a carriage return to advance to the next location.

20.4.2 Backing Up Through Files

When you type the up-arrow character (^) followed by a carriage return in place of entering data into the current location, SIPP closes the current location and opens the previous location. If you specify a value followed by an up-arrow, SIPP enters that value into the current location, closes that location, and opens that location.

If you type an up-arrow when the offset from a specified base is 0, SIPP opens the previous location and displays the offset as a double-precision negative number.

In the following example, the value 112000 is entered into the current location, and the previous location is opened.

```
Base        Offset        Old    New?
 002000      002134    020027    112000^RET
 002000      002132    001732
```

In the last example, notice how SIPP decrements the offset by 2 to designate the previous 16-bit location.

You can use the up-arrow after you use the backslash (\) to back up to the previous byte (if currently in word mode). You can also use the up-arrow after you use the slash (/) to back up to the previous word (if currently in word mode).

20.4.3 Advancing in Bytes

By default, SIPP operates in word mode. That is, locations are displayed as 16-bit words, and values are displayed and entered as word values. If you type the backslash character (\), SIPP closes the current location, and reopens the low, or even byte, of that location. Values that are displayed and entered from this point are in bytes.

To revert to word mode, type the slash character (/). When you type the slash, SIPP reopens the current location as a 16-bit word.

The following example uses the backslash to advance in byte mode, and then the slash to revert to word mode. Notice that SIPP prints out a new header each time it changes from word to byte mode, and vice versa.

```
Base        Offset        Old    New?
 0020000     002112    003002
                                 \RET
Base        Offset        Old    New?
 002000      002112    002
 002000      002113    006       RET
 002000      002114    132       RET
                                 /RET
Base        Offset        Old    New?
 002000      002114    003132
```

If you are in byte mode when you get the Offset? prompt, SIPP automatically resets itself to word mode.

20.4.4 Entering Octal Values (;O)

Use the ;O command, followed by a carriage return, to reopen the current location, and display its contents as an octal value. Since octal mode is the default setting, you need to use it only if you are currently operating in ASCII or Radix–50 mode and wish to revert to octal mode. You can also use the ;O command to switch from byte mode to word mode.

The following example uses the ;O command to switch from ASCII mode to octal mode.

```
Base        Offset        Old    New?
 002000       002100   051101
 002000       002100      <A>     ;A RET
                                  ;O RET
Base        Offset        Old    New?
 002000       002100   051101
```

Note that unlike the ;A and ;R commands, ;O accepts no optional argument. If you return to the Offset? prompt, SIPP automatically resets itself to octal mode.

20.4.5 Displaying and Entering ASCII Values

Use the ;A command, followed by a carriage return, to open the current location as a byte and display its contents as an ASCII value. When you use the ;A command, SIPP continues to display contents in ASCII until you use the ;O or ;R command. Note that when you operate in ASCII mode, you advance through the file in byte mode.

The following example uses the ;A command to open the low byte of the current location and display its contents as an ASCII value.

```
Base        Offset        Old    New?
 003000      003100    050524   ;A(RET)
 003000      003100       <T>   (RET)
 003000      003101       <Q>
```

Use the ;Ax command to insert an ASCII character, represented by x, into the low byte of the current location. When you use the ;Ax command, SIPP enters the ASCII character directly into the current byte, closes that byte, opens and displays the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ax command). Note that you can insert only one ASCII character at a time, and that you should not insert control characters. You can use the ;Ax command when displaying in ASCII mode.

The next example uses the ;Ax command to enter the ASCII character, W, into the current byte and proceed to the next byte.

```
Base        Offset        Old    New?
 003000      003100    050524   ;AW(RET)
 003000      003101       121
```

You can also use the ;Ax command to search for an ASCII value (see Section 20.4.7).

20.4.6 Displaying and Entering Radix–50 Values

Use the ;R command, followed by a carriage return, to reopen the current location and display its contents in Radix–50. When you use the ;R command, SIPP continues in Radix–50 mode until you use either the ;A or ;O command. Note that Radix–50 mode advances in word mode.

The following example uses the ;R command to reopen the current location and display its contents as a Radix–50 value.

```
Base        Offset        Old    New?
 001000      005220    071070   ;R(RET)
 001000      005220      <RK>   (RET)
 001000      005222     <TES>
```

If the contents of a location is an invalid Radix–50 value, SIPP displays the contents as <???>

You can use the ;Ryyy command to insert up to three Radix–50 characters, represented by yyy, into the current location. When you use the ;Ryyy command, SIPP inserts the Radix–50 value into the current location, closes the current location, and opens and displays the contents of the next location as an octal, ASCII, or Radix–50 value (depending on what mode you were in prior to using the ;Ryyy command).

If you use the ;Ryyy command, and you enter only two Radix–50 characters, SIPP inserts a blank as the third character. Likewise, if you enter only one Radix–50 character, SIPP inserts blanks for the second and third characters. If you use an imbedded blank (for example, X Z), SIPP inserts all characters as typed. Note that you can insert up to only three Radix–50 characters at a time. Use the ;Ryyy command only when the low byte of the current location is open; SIPP prints an error message if you attempt to insert a Radix–50 value when the high byte of the current location is open.

The following Radix–50 values are valid for use with the ;Ryyy command:

A through Z
0 through 9
$
*
.
%

Note that a space is also a valid Radix–50 character, and that SIPP translates the percentage character to a dot (.).

The following example uses the ;Ryyy command to insert three Radix–50 characters in the current location, and proceed to the next location.

```
Base        Offset        Old    New?
 001000      005332    000240    ;RABC(RET)
 001000      005334    002110
```

You can also use the ;Ryyy command to search for a Radix–50 value (see Section 20.4.7).

20.4.7 Searching Through Files (;S)

You can use the ;S command to search between two specified boundaries of a file for a given value. With this feature, you can find the location where you want to make a change by searching for a specific value.

To request a search, type the following in response to any of SIPP's dialog questions or in place of entering new data into the current location.

```
;S
```

Do not type the ;S command in response to the Checksum? prompt. After you type the ;S command, SIPP responds with the following prompt:

```
Search for?
```

Enter the value for which you want SIPP to search. You can use the ;Ax or ;Ryyy command in response to the last prompt to search for ASCII or Radix–50 values. If you type a backslash after the value you enter, SIPP searches for a byte value. Otherwise, it searches for a word value. Note that if you use the ;Ax notation to search for an ASCII value, SIPP conducts the search in byte mode.

SIPP then asks for the lower address limit at which to begin the search:

```
Start?
```

Enter an address, followed by a carriage return, or just a carriage return. If you enter a carriage return, SIPP begins its search at the beginning of the file. If you enter an address, SIPP begins the search at that address. If you are searching through an overlay segment, use the following notation for the start address:

```
n:m
```

In the n:m notation, n represents the number of the segment you want to search, and m represents the offset from the start of the segment where you want SIPP to begin the search.

SIPP then asks for the upper address limit for the search:

```
End?
```

You can enter an address (including the n:m notation) or a carriage return. If you enter a carriage return, SIPP searches to the end of the file or volume. (If you use the /A option, SIPP searches to the end of the last block in the file or volume; otherwise it searches up to and including the last address in the program.) If you enter an address, SIPP conducts the search up to, but not including, that address.

After you have specified the search limits, the search begins. Each time SIPP finds a value that matches the one you specified, SIPP prints out the address of that value. If you use the /A option or if SIPP is searching the root segment of a program, SIPP prints the search results as follows:

```
Found at nnnnnn
```

If a search crosses segments in an overlaid file, SIPP prints the following each time it finds the specified value:

```
Found at seg:mmm,nnn
```

In the seg:mmm,nnn notation, seg represents the segment number, mmm represents the load map address of the segment, and nnn represents the offset from the start of the specified segment. Note that if you are searching an overlaid file, and you have specified the /A option in the command line, SIPP does not use the seg:mmm,nnn notation.

20.4.8 Verifying (;V)

Use the ;V command to list at the terminal all the changes you have made during the current patching session. After SIPP prints out the addresses and new contents of all the locations that have changed, SIPP returns you to the operation that was interrupted.

You can use the ;V command at any time, except in response to the Checksum? prompt and the search Start?, and End? prompts. You can use the ;V command in response to the Search for? prompt.

The following example uses the ;V command to list at the terminal all the changes that have been made during the current patching session.

```
Base        Offset        Old    New
 003000      003200    003112    240RET
 003000      003202    002300    RET
 003000      003204    002300    RET
 003000      003206    000230    240RET
 003000      003210    000101    ;VRET

 Base       Offset        Old    New
 003000      003200    003112    000240
 003000      003206    000230    000240

 Base       Offset        Old    New
 003000      003210    000101
```

Note that when you use the ;V command to verify your modifications, all displays are in octal words. Note also that if you change a location and later restore that location to its original contents, SIPP includes that location in the verification.

20.4.9 Backing Up to a Previous Prompt

You can use the CTRL/Z sequence (or up-arrow Z), followed by a carriage return, to back up to a previous prompt. For example, after you have modified a series of locations, you can type CTRL/Z, followed by a carriage return, to back up to the Offset? prompt. Backing up to a previous prompt enables you to examine and/or modify other series of locations in your program.

If you use CTRL/Z in place of entering a value into a location, SIPP prompts Offset?. If you type CTRL/Z, followed by a carriage return, in response to Offset?, SIPP prompts Base?. If you type yet another CTRL/Z, followed by a carriage return, SIPP either:

1. Prompts Segment?, if the file is overlaid, or
2. Prompts you for a checksum (if you used /C), then installs the patch (if the checksum is valid). (Note that if you have used the /L option, SIPP does not install the patch, but does create the command file, if requested.)

If the file is overlaid, and you type another CTRL/Z followed by a carriage return sequence in response to Segment?, SIPP prompts you for a checksum (if specified) then installs the modifications.

Using CTRL/Y provides a more efficient way of installing a patch (see the following subsection). The CTRL/Z sequence is designed primarily to request a particular prompt.

20.4.10 Completing Code Modifications

You can type the CTRL/Y sequence (or up-arrow Y), followed by a carriage return, to install the code modifications you have entered. If you have used the /C option, which requires you to enter a checksum, SIPP will prompt you for a checksum before it installs the modifications. If the checksum you type is valid, SIPP then installs the patch. If you have used the /L option and you enter the correct checksum, SIPP does not install the patch, but does create the command file, if requested.

After SIPP installs the modifications, an asterisk appears in the left margin, indicating that SIPP is ready to accept a new command string.

20.4.11 Extending Files and Overlay Segments

The limits to which you can extend programs and overlay segments while patching vary, depending on if the program is

- Nonoverlaid
- Overlaid, but has only low memory overlays
- Overlaid, but has only extended memory overlays
- Overlaid, and has both low memory and extended memory overlays

The subsections that follow describe in detail the restrictions on extending programs, root sections, and overlay segments. Each subsection also details what data within your program SIPP does or does not automatically modify as you extend root sections and/or overlay segments.

Listed below are the data that SIPP may automatically modify as you make extensions. (Note that each is identified by an abbreviation; the subsections that follow reference these data by their abbreviations.)

Location 50	Contains the last address used by the program, if the program is nonoverlaid. If the program has low memory overlays, location 50 contains the last address used by the low memory overlay region(s). If the program has only extended memory overlays, location 50 contains the last address used by the root.
Reg. Size	Indicates the size of the extended memory region. This data appears in the extended memory overlay handler.
High Root + 2	Indicates the address of the next available location beyond the root segment. This data appears in either the low memory overlay handler or the extended memory handler.

High /O + 2	Indicates the address of the next available location beyond the last low memory overlay region. This data appears in either the low memory overlay handler or the extended memory overlay handler.
Wdcnt. in Seg.	Indicates the number of words in the current overlay segment. This data appears in the overlay handler segment table.
WDB Size and Length	Indicates the window size and length to map in the window definition block (WDB) for the extended memory overlay you are extending. This data appears in each extended memory overlay segment's WDB.
WDB Offset	Indicates the offset into the extended memory region of the windows following the segment you are extending. This data appears in each extended memory overlay segment's WDB.

20.4.11.1 Nonoverlaid Program – If necessary, SIPP automatically extends a nonoverlaid program up to the end of the last block of the save image on which the program exists (SIPP automatically modifies location 50). Refer to DUP (Chapter 6) for details on extending a nonoverlaid program beyond that point.

20.4.11.2 Overlaid Program, Low Memory Overlays Only – If your program is overlaid, but has only low memory overlays, SIPP does not permit you to extend the root. If an overlay segment is not in the last overlay region, you can extend it to the size of the largest segment in its region. If the overlay segment is in the last overlay region, you can extend it to the end of the last block of that overlay segment.

Table 20–3 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has low memory overlays only. Note in this table that there is a column heading for an overlay segment that is not the largest in its overlay region (Not Largest in Region), and for an overlay segment that you extend beyond the largest overlay segment in its region (Past Largest in Last Region). YES indicates that SIPP does modify the data in question if necessary, NO indicates SIPP does not modify the data in question, and N/A indicates that the data in question is not applicable.

20.4.11.3 Overlaid Program, Extended Memory Overlays Only – If your program is overlaid, but has extended memory overlays only, you can extend the root segment up to the end of the last block on which the root resides. You can also extend any overlay segment up to the end of the last block of that particular segment, so long as you do not exceed the physical address space.

Table 20–3: Overlaid Program Segment Limits

	Not Largest in Region	Past Largest in Last Region
Location 50	NO	YES
Reg. Size	N/A	N/A
High Root + 2	NO	NO
High /O + 2	NO	YES
Wdcnt. in Seg.	YES	YES
WDB Size and Length	N/A	N/A
WDB Offset	N/A	N/A

Table 20–4 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has extended memory overlays only. Note in this table that there is a column heading for the root (Root), an overlay segment that is not the largest in its overlay region (Not Largest), and an overlay segment that you extend beyond the largest overlay segment in its region (Past Largest). YES indicates that SIPP does modify the data in question if necessary, and NO indicates SIPP does not modify the data in question.

Table 20–4: Overlaid Program Segment Limits

	Root	Not Largest	Past Largest
Location 50	YES	NO	NO
Reg. Size	NO	YES	YES
High Root + 2	YES	NO	NO
High /O + 2	YES	NO	NO
Wdcnt. in Seg.	NO	YES	YES
WDB Size and Length	NO	YES	YES
WDB Offset	NO	YES	YES

20.4.11.4 Overlaid Program, Both Low Memory and Extended Memory Overlays – If your program has both low memory and extended memory overlays, SIPP does not permit you to extend the root segment. You can extend any low memory overlay segment up to the size of the largest segment in the same region. If the low memory overlay segment is in the last low memory overlay region, you can extend it to the end of the last block of that overlay segment.

You can extend any extended memory overlay segment up to the end of the block limit of that particular segment, so long as you do not exceed your physical address space.

Table 20–5 shows the locations that SIPP modifies if necessary when you extend the root or overlay segment of a program that has both low memory and extended memory overlays. Note in this table that there is a column heading for a low memory overlay segment that is not the largest in its overlay region (/O Not Largest), a low memory overlay segment that extends beyond the largest segment in its region (/O Past Largest), an extended memory overlay segment that is not the largest in its region (/V Not Largest), and an extended memory overlay segment that extends beyond the largest segment in its region (/V Past Largest). YES indicates that SIPP does modify the data in question if necessary, and NO indicates SIPP does not modify the data in question.

Table 20–5: Overlaid Program Segment Limits

	/O Not Largest	/O Past Largest	/V Not Largest	/V Past Largest
Location 50	NO	YES	NO	NO
Reg. Size	NO	NO	YES	YES
High Root + 2	NO	NO	NO	NO
High /O + 2	NO	YES	NO	NO
Wdcnt. in Seg.	YES	YES	YES	YES
WDB Size and Length	NO	NO	YES	YES
WDB Offset	NO	NO	YES	YES

NOTE

It is possible when extending extended memory overlay segments to exceed the 96K word physical address space (SIPP prints a warning message if you do this). If you do exceed the 96K word limit, use the CTRL/C sequence to abort the patching session; many system communication area locations will have already changed to contain invalid data.

20.5 SIPP Checksum

SIPP's checksum algorithm creates the checksum only after you have finished creating a patch. The checksum helps you verify your work. It lets you compare the patch you make to another that is known to be correct. The checksum does not tell you where your error is, but it does tell you that an inconsistency exists. SIPP's checksum algorithm uses the calculated address of a changed location and its new contents. SIPP includes in its checksum only those values of locations that have changed during a patching session.

If you change a word several times during a patching session, SIPP enters into its checksum only the last value you specify. This feature allows you to correct a mistake, yet maintain a valid checksum.

If you are creating a checksum (/D option), SIPP prints the following when you finish the patch:

```
Checksum=nnnnnn
```

If you are verifying a checksum (/C option), SIPP asks the following when you complete the patch:

```
Checksum?
```

Respond by entering the checksum for the patch.

If the checksum is incorrect, SIPP prints:

```
?SIPP-E-Checksum error
```

SIPP then returns to the beginning of its dialog (the Segment? or Base? prompt), allowing you to find and correct your error without exiting from the patching session. In this way, you do not lose the changes you have made; you can go back and verify them (see Section 20.4.8 for details on the verify command).

SIPP does not install the patch until you enter the correct checksum. You can type CTRL/C to abort the patching procedure.

20.6 Running SIPP from an Indirect File

The SIPP indirect command file contains the commands necessary to install a patch in a particular file or volume. The order in which the modifications appear in the command file may not correspond to the actual sequence in which you typed them; however, the changes are the same as you typed. The contents of the command file always appear as octal word values. When you specify a command file in the initial command string, SIPP creates that file for use as an indirect command file. If you use the /L option when you create the command file, SIPP installs the patch contained within only when you run this file as an indirect command file. By default, SIPP assigns this file a .COM default file type.

A command file always contains a checksum generated during the console input session. If you use the /C option, SIPP prompts you for a checksum after you finish making the code modifications. If the checksum is valid, SIPP completes this command file, and you can execute this command file at any time you wish. If you use the /A option, SIPP inserts /A in the command file.

The command file TEST.COM is created in the following example. (Note that SIPP does not modify TEST.SAV in the example, because /L was specified in the command string.)

```
. R SIPP(RET)
* TEST=TEST/L(RET)
Base?    5000(RET)
Offset?  20(RET)

Base        Offset        Old     New
 005000      000020     032764     240(RET)
 005000      000022     177400     240(RET)
 005000      000024     000002     1016(RET)
 005000      000026     001016     (CTRL/Y)(RET)
 *,(CTRL/C)
```

A copy of TEST.COM as it appears in indirect command file format follows.

```
. TYPE TEST.COM
RUN SIPP
DK:TEST.SAV/C
5000
20
240
240
1016
^Y
165617
^C
```

The number 165617 (the last line in the file) is the checksum for that patch.

To run the command file TEST.COM as an indirect file, type the following in response to the monitor dot.

```
@TEST(RET)
```

If you run a SIPP indirect command file when the SET TT: QUIET setting is in effect, SIPP overstrikes its output at the terminal but does install the patch correctly.

20.7 Running SIPP from a BATCH Stream

An easy way to install a patch from a BATCH stream is to follow the instructions for creating a command file. When you get your command file, simply open it with an editor, enter the BATCH commands, and insert a dot before the line RUN SIPP, and insert asterisks before each subsequent line. Remember to remove the CTRL/C (^C) from the command file. An example of preparing TEST.COM (from the previous section) for a BATCH stream follows.

```
$JOB/RT11
        TTYIO
.RUN SIPP
*DK:TEST.SAV/C
*5000
*20
*240
*240
*1016
*^Y
*165617
$EOJ
```

Chapter 21
Source Language Patch Program (SLP)

The source language patch program (SLP) is a patching tool you can use for maintaining source files that exist on any RT–11 device.

SLP accepts as input a source file you wish to patch and a command file that you create when you compare two source programs using the source compare program, SRCCOM, described in Chapter 15. When you use SLP along with the SRCCOM command file, you can quickly and easily patch one version of a source program to match another version.

Chapter 15, Source Compare Program (SRCCOM), describes the procedure you can use to create a patch command file that is suitable for input to SLP.

21.1 Calling and Terminating SLP

To call SLP from the system device, type the following in response to the keyboard monitor dot (.):

```
. R SLP(RET)
```

The Command String Interpreter (CSI) prints an asterisk (*) at the left margin of the terminal and waits for a command string. If you enter only a carriage return in response to the asterisk, SLP prints its current version number. You can type CTRL/C to halt SLP and return control to the monitor when SLP is waiting for input from the console terminal. To restart SLP, type R SLP or REENTER in response to the monitor's dot.

21.2 SLP Command String Syntax

Chapter 1, Command String Interpreter, describes the general syntax of the command line that SLP accepts.

Enter a command line according to this general syntax:

[outfil][,listfil] = infil,comfil/[option...]

where:

outfil — represents the updated source file. The default file type is .MAC.

.listfil — represents the listing file. When you specify this file, SLP creates a numbered listing of the updates SLP made to the source file. The default file type is .LST.

infil — represents the source file you want SLP to update. The default file type is .MAC.

comfil — represents the command file that contains the commands for updating the source file. The default file type is .MAC. You can create this file by specifying a SLP-filespec in a SRCCOM command line. A SLP input file created by SRCCOM has the file type .SLP.

/option — represents one of the options listed in Table 21–1.

Although either output files can be omitted, you must use one or both.

21.3 Options

Table 21–1 lists the options you can use in the command line.

21.4 Example

This section uses SLP to patch the source file, ANTONY.MAC, so that it matches the source file CAESAR.MAC. CAESAR.MAC consists of the following lines.

```
FRIENDS, ROMANS, COUNTRYMEN!
LEND ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MEN DO
LIVES AFTER THEM.
THE GOOD IS OFT INTERRED
WITH THEIR BONES;
SO LET IT BE WITH CAESAR!
```

The file this example will patch, ANTONY.MAC, follows.

```
FRIENDS, ROMANS, COUNTRYMEN!
LEN ME YOUR EARS!
I COME TO BURY CAESAR,
NOT TO PRAISE HIM.
THE EVIL THAT MAN DO
LIVES AFTER THEM.
THE GOOD IS OFT ENTERED
WIT THEIR HOMES;
SO LET IT BE WITH CAESAR!
```

Table 21–1: SLP Options

Option	Function
/A	Disables audit trail generation. The audit trail is a string of characters that SLP appends to the end of each updated line in the output files. The audit trail keeps track of the update status of each line in the output file. You can use the /A option if you do not want SLP to use the audit trail in both the updated source file and the listing file.
/B	Inserts spaces instead of tabs between the source line and the audit trail.
/C[:n]	Determines or validates the contents of the SLP input file, SLP command file, or both by using checksums. Use /C to determine the checksum of a file. Use /C:n to verify the contents of a file. SLP computes the checksum for the file, and compares the result to the value you specify with n.
/D	Creates a double-spaced listing. When you use this option, SLP double-spaces between the lines in a listing file.
/L:n	Specifies the size of the source line, where n represents the maximum number of characters you want in the source line. The default buffer size for formatting lines is 200(decimal) bytes. If you expect the size of the command lines or source lines to be greater than what can fit in the line buffer, you can use this option to change the buffer size. SLP interprets the number you specify for n as an octal number; if you enter a decimal number, use a decimal point. The line buffer must be at least as long as the sum of the column number where the audit trail begins and the number of characters in the audit trail.
/N	Suppresses the creation of a backup file when SLP updates the input file.
/P:n	Specifies the start column of the audit trail, where n represents the column number in which you want the audit trail to start. If the number you specify for n is decimal, be sure to use a decimal point after the number. By default, SLP starts the audit trail in column 73 (decimal). If a source line extends beyond the column where the audit trail begins, the audit trail can overstrike the source line. If you use the /P:n option, you start the audit trail in any tab stop column. SLP rounds up the number you specify to the nearest tab stop column. If, for example, you specify 46 for n, SLP rounds this number to 49.
/S:n	Specifies size of the audit trail, where n represents the number of characters you want in the audit trail. If the number you specify is decimal, be sure to use a decimal point after the number. The default number of characters in the audit trail is 12(decimal). The maximum number of characters you can specify for the audit trail is 16(decimal).
/T	Retains trailing blanks and tabs in the input source file. By default, SLP removes spaces and tabs that appear at the end of lines in the input source file.

By specifying a SLP-filespec in a SRCCOM command line, this example obtains a command file, CAESAR.SLP. CAESAR.SLP contains the necessary commands to make ANTONY.MAC match CAESAR.MAC. The following command line directs SLP to patch ANTONY.MAC so that it matches CAESAR.MAC.

```
. R SLP
* ANTONY,ANTONY=ANTONY,CAESAR.SLP
```

After executing the command above, SLP assigns a .BAK file type to the input file ANTONY.MAC. It assigns .MAC file type to the updated source file. SLP has also created a listing of ANTONY.MAC that lists each line by number and appends an audit trail to each new line. The updated file, ANTONY.MAC, is now identical to CAESAR.MAC. ANTONY.LST appears below.

```
SLP V05.00                    ANTONY,ANTONY=ANTONY.MAC,CAESAR.SLP

     1.  FRIENDS, ROMANS, COUNTRYMEN!          ;**NEW**
     2.  LEND ME YOUR EARS!                    ;**-1
     3.  I COME TO BURY CAESAR,
     4.  NOT TO PRAISE HIM.                    ;**NEW**
     5.  THE EVIL THAT MEN DO                  ;**-1
     6.  LIVES AFTER THEM.                     ;**NEW**
     7.  THE GOOD IS OFT INTERRED              ;**NEW**
     8.  WITH THEIR BONES;                     ;**-2
     9.  SO LET IT BE WITH CAESAR!
```

Note that when SLP updates a line, it appends an additional audit trail below the audit trail of the updated line. The additional audit trail keeps track of the number of consecutive lines that have been updated. In ANTONY.LST, above, note the audit trails ;**–1 and ;**–2.

21.5 Creating and Maintaining a Command File

SLP is a line-oriented patching tool. That is, you make changes to entire lines, and not to individual characters or strings of characters within a line. If you want to change only a few characters within a line, it will be necessary for you to enter a new line.

Although DIGITAL recommends that you create the SLP input command file by specifying a SLP-filespec in a SRCCOM command line, you can use any RT–11 editor to create it yourself. The section that follows describes the commands, or operators, you use to create the command file. This procedure is tedious, however, and in most cases unnecessary. But for completeness, this procedure is included with this chapter. Table 21–2 lists the commands, or operators, you enter into the command file.

The section ends with a description of various line manipulations that SLP can effect.

21.5.1 Update Line Format

The general format of the SLP command file update line follows.

```
-locator1,[locator2],[/audit trail/][;]
inputline
    .
    .
    .
```

Table 21–2: SLP Command File Operators

Operator	Function
-	Indicates the start of an update. SLP ignores any data that precedes this operator in a SLP command file. If SLP finds characters before this operator in a command file, SLP prints a warning and the characters are ignored. If no operator of this type is found in a command file, SLP prints an error message and the CSI prompt (*) appears.
\	Disables the audit trail. Note that this operator must appear on a line by itself. If it appears in the first column of a line with additional information following it, the audit trail will be disabled, but the rest of the command line will be ignored. If used in any column other than column one, a syntax error occurs.
%	Enables the audit trail.
/	Indicates the end of an update or a series of updates; it appears as the last character in the command file.
//	Indicates the end of one of a series of update texts in a single command file; each text updates one input file. This operator is used when you want to include updates for more than one file in a single command file. Type the double slash (//) on a line by itself after each update text in the series. Then type on the next line the command line that specifies the next input file to be updated, and on the succeeding lines the update text for that file. Note that the command file specified in each command line must be the same as the command file specified on the first command line.
<	Serves as an escape character for characters SLP would otherwise interpret as operators. For example, if you want to include a slash (/) in a source file, type the less-than character (<) before the slash. Then, SLP will not interpret the slash as an operator. You can use the less-than character as an escape character for all SLP command file operators.

where:

-	indicates that this is an update line.
locator1	represents a character string that serves as a line locator. SLP moves the line pointer to the line specified by the line locator. You can specify this line locator with any of the locator forms described below.
locator2	represents a character string that, when used with locator1, defines the end of a range of lines you want to delete or replace. You can specify this line locator with any of the locator forms described below. You cannot define a range of lines in a backwards direction; the line referenced by locator2 must occur in the source file after the line referenced by locator1.
/audit trail/	represents a character string you use as an audit trail. SLP appends the audit trail to the right of each updated line. You must delimit the audit trail with slashes (/).

inputline	represents a line of new text that SLP inserts into the file immediately following the current line. You can enter as many input lines as you want.
;	is an optional command line terminator.

All fields in the update line are positional. That is, if you specify only locator1 and an audit trail, you must use two commas between those two fields. If you want to specify only the audit trail, you must precede the audit trail with two commas.

The update lines in a command file must edit the source file in a forwards direction, from beginning to end. Each locator1 must point to a line that appears in the source file before the lines pointed to by any succeeding locator1.

The line locators can take one of the following forms:

```
/string/[+n]
/string...string/[+n]
number[+n]
 +n
```

where:

/string/[+n]	represents an ASCII character string. You must delimit any string you enter with slashes. SLP locates the first occurrence of this string, and moves the line pointer to the line that contains that string. +n represents the offset from the line that contains the string. You must use the plus character (+) with the n notation.
/string...string/[+n]	represents an ASCII character string. SLP locates the line in which the two strings delimit a larger string. Use the ellipsis (...) in this locator form to separate the two strings. +n represents the offset from the line specified by the string...string locator.
number[+n]	represents the line number to which SLP is to move the line pointer. +n represents the offset from the line specified by number.
+n	represents the offset from the current line pointer. SLP interprets the period (.) as the current line pointer location, and the +n as the offset from it. You must use the plus character (+).

21.5.2 Creating a Numbered Listing

You can use SLP to create a numbered listing of the input source file. In creating a command file, you should use a numbered listing when you prepare command input. To generate a numbered listing, enter the following lines:

```
.R SLP
*,listfile=infile
```

Listfile represents the listing file SLP produces, and infile represents the input source file. Here is a file, PROG.MAC, from which SLP is to create a numbered listing:

```
        .TITLE   PROG.MAC    VERSION 1
        .MCALL   .TTYOUT, .EXIT, .PRINT

EXP:    .PRINT   #MESSAG
        MOV      #N,R5
FIRST:  MOV      #N+1,R0
        MOV      #A,R1
```

The following command line creates a numbered listing, PROG.LST of the file above, PROG.MAC:

```
* ,PROG=PROG
```

After SLP processes the command above, it produces the following listing of PROG.MAC:

```
SLP  V05.00                          ,PROG=PROG.MAC

     1.            .TITLE   PROG.MAC    VERSION 1
     2.
     3.            .MCALL   .TTYOUT, .EXIT, .PRINT
     4.
     5.   EXP:     .PRINT   #MESSAG
     6.            MOV      #N,R5
     7.   FIRST:   MOV      #N+1,R0
     8.            MOV      #A,R1
```

21.5.3 Adding Lines to a File

To add lines to a file, enter in the command file one of the three locator forms below:

-number

-.[+n],,[/audittrail/]

-/string/

Notice in the second locator form the two commas between the locator and the audit trail. You do not have to insert these commas if you are not specifying an audit trail.

Below is a file, NUMBER.PAS, to which SLP is to add new lines.

```
PROGRAM NUMBER;

TYPE    TEXT     =FILE OF CHAR;
        PTR      =^WORDNODE;
        WORDNODE=RECORD
                  WORD:ARRAY[1..30] OF CHAR;
                  NEXT:PTR;
                  END;

VAR     P,TOP    :PTR;
        INTEXT   :TEXT;
        I        :INTEGER;
```

SLP is to insert the following line between the fourth and fifth lines of NUMBER.PAS.

The command file, OMEGA.MAC, contains the following update 1 to perform this procedure.

```
-/PTR/
(*POINTER TO NODE*)
/
```

When SLP processes OMEGA.MAC with NUMBER.PAS, it produces the following updated listing file.

```
SLP V05.00                NUMBER.PAS,NUMBER=NUMBER.PAS,OMEGA.MAC

 1. PROGRAM NUMBER;
 2.
 3. TYPE      TEXT     =FILE OF CHAR;
 4.           PTR      =^WORDNODE;
 5. (*POINTER TO NODE*)                                  ;**NEW**
 6.           WORDNODE=RECORD
 7.                     WORD:ARRAY[1..30] OF CHAR;
 8.                     NEXT:PTR;
 9.                     END;
10.
11. VAR       P,TOP    :PTR;
12.           INTEXT   :TEXT;
13.           I        :INTEGER;
```

SLP has numbered the lines, inserted the new text, and appended the default audit trail (;**NEW**) to the new line.

The next example uses the same source file, but uses this command in the command file, SIGMA.MAC:

```
-/WORDNODE=/+2
                ID   :INTEGER;
/
```

When SLP processes SIGMA.MAC with the source file NUMBER.PAS, it generates the following listing file:

```
SLP V05.00                NUMBER.PAS,NUMBER=NUMBER.PAS,SIGMA.MAC

 1. PROGRAM NUMBER;
 2.
 3. TYPE      TEXT     =FILE OF CHAR;
 4.           PTR      =^WORDNODE;
 5.           WORDNODE=RECORD
 6.                     WORD:ARRAY[1..30] OF CHAR;
 7.                     NEXT:PTR;
 8.                     ID   :INTEGER;
 9.                     END;                            ;**NEW**
10.
11. VAR       P,TOP    :PTR;
12.           INTEXT   :TEXT;
13.           I        :INTEGER;
```

Again, SLP has numbered the lines, and this time it skips two lines after the first occurrence of string WORDNODE before inserting the new input line.

You can include in one command file update text for several input files. Type a double slash (//) on a line by itself at the end of the update text for each file. Begin the update text for the next file with a line containing only the command line that specifies the input file to be updated by the next text. Then type the update text on the lines that follow the command line. Type a slash (/) at the end of the command file.

For example, the command file MTST.MAC contains update text to patch the files NUMBER.PAS and GTMSG.MAC, in that order.

```
-/WORDNODE=/+2
                ID   :INTEGER;
//
DY0:NEWMSG=GTMSG.MAC,MTST.MAC
-2,2
    .IDENT    /01.01/
-7
    ADD       A,B
-14
B:  .WORD     0
/
```

21.5.4 Deleting Lines in a File

The SLP command file command syntax for deleting lines from a file is:

-locator1,locator2,[/audittrail/][;]

where locator1 and locator2 can be any of the forms of the locator fields described earlier. locator1 specifies the line where SLP is to begin deleting lines. locator2 specifies the last line SLP is to delete.

If you want to delete lines five through eight in file NUMBER.PAS, it will be helpful to look at a numbered listing of NUMBER.PAS.

```
SLP V05.00                                        ,NUMBER=NUMBER.PAS

 1. PROGRAM NUMBER;
 2.
 3. TYPE      TEXT     =FILE OF CHAR;
 4.           PTR      =^WORDNODE;
 5.           WORDNODE=RECORD
 6.                     WORD:ARRAY[1..30] OF CHAR;
 7.                     NEXT:PTR;
 8.                     END;
 9.
10. VAR       P,TOP    :PTR;
11.           INTEXT   :TEXT;
12.           I        :INTEGER;
```

In the command file, GAMMA.MAC, the command for deleting lines five through eight follows.

```
-/WORDNODE=/,/END/
/
```

When SLP processes GAMMA.MAC with NUMBER.PAS, it produces this listing file of NUMBER.PAS.

```
SLP V05.00                NUMBER.PAS,NUMBER=NUMBER.PAS,GAMMA.MAC

1.   PROGRAM NUMBER;
2.
3.   TYPE      TEXT     =FILE OF CHAR;
4.             PTR      =^WORDNODE;
5.                                                        ;**-4
6.   VAR       P,TOP    :PTR;
7.             INTEXT   :TEXT;
8.             I        :INTEGER;
```

21.5.5 Replacing Lines in a File

When you replace lines, you delete and then add new text. To replace lines in a file, first enter the full SLP edit command for the delete operation. The first line locator specifies the first line to be deleted. The second line locator specifies both the last line to be deleted and the location where SLP is to insert new text. For example, the command file command instructs SLP to move the line pointer to line 4.

```
-4,.+4
```

Then, SLP is to delete the next four lines (represented by +4), including line 4. Finally, SLP is to insert input lines that follow in the command file. SLP inserts the new lines, beginning at the line pointer's current location.

The following example illustrates replacing lines in a file. The source file, BETA.MAC, consists of the following lines:

```
        .TITLE   BETA.MAC
        .MCALL   .TTYOUT, .PRINT, .EXIT
START:  .PRINT  #MESSAG
        MOV     #5,R0
```

The command file, DELTA.MAC, contains:

```
-6,6,/;AUDIT TRAIL/
BNE          START:
MOVB (R2),-(R3)
/
```

When SLP processes DELTA.MAC with BETA.MAC, it produces the following listing file:

```
SLP V05.00   BETA,BETA=BETA,DELTA.MAC

1.           .TITLE   BETA.MAC
2.
3.           .MCALL   .TTYOUT, .PRINT, .EXIT
4.
5.   START:  .PRINT  #MESSAG
6.           BNE     START:                  ;AUDIT TRAIL
7.           MOVB     (R2),-(R3)             ;AUDIT TRAIL
```

21.5.6 Determining and Validating the Contents of a File

Use the checksum option (/C[:n]) to determine or validate the contents of a file. The checksum option directs SLP to compute the sum of all ASCII data in a file. If you specify the command in the form /C:n, /C directs SLP to compute the checksum and compare that checksum to the value you specify as n.

To determine the checksum of a file, enter the SLP command line with the /C option applied to the appropriate file (the file whose checksum you want to determine). For example, SLP responds to the command

```
INFILE,INFILE=INFILE.MAC/C,INFILE.SLP
```

with the message

```
?SLP-I-DEV:FILNAM.TYP checksum is n
```

SLP generates a similar message when you request the checksum for the command file.

To validate the changes made to a file, enter the checksum option in the form /C:n. SLP compares the value it computes for the checksum with the value you specify as n. If the two values do not match, SLP enters no changes and displays a message reporting the checksum error as either a source file or a correction file checksum error, whichever is appropriate.

```
?SLP-F-Source file checksum error
```

or

```
?SLP-F-Correction file checksum error
```

Checksum processing always results in a nonzero value.

Do not confuse this checksum with the record checksum byte.

Appendix A
BATCH

RT–11 BATCH is a complete job control language that allows RT–11 to operate unattended. BATCH processing is ideally suited to frequently run production jobs, large and long-running programs, and programs that require little or no interaction with you, the user. With BATCH, you can prepare your job on any RT–11 input device and leave it for the operator to start and run.

RT–11 BATCH permits you to:

- Execute an RT–11 BATCH stream from any RT–11 input device
- Output a log file to any RT–11 output device (except magtape or cassette)
- Execute the BATCH stream with the SJ monitor or in the background with the FB monitor or XM monitor
- Generate and support system-independent BATCH language jobs
- Execute RT–11 monitor commands from the BATCH stream

RT–11 BATCH consists of the BATCH compiler and the BATCH run-time handler. The BATCH compiler reads the batch input stream you create, translates it into a format suitable for the RT–11 BATCH run-time handler, and stores it in a file. The BATCH run-time handler executes this file with the RT–11 monitor. As each command in the batch stream executes, BATCH lists the command, along with any terminal output generated, by executing the command on the BATCH log device.

A.1 Hardware and Software Requirements

You can run RT–11 BATCH on any single-job foreground/background or extended memory system that is configured with at least 16K words of memory. A line printer, although optional, is highly desirable as the log device.

BATCH uses certain RT–11 system programs to perform its operations. For example, the $BASIC command executes the file BASIC.SAV. Make sure that the following RT–11 programs are on the system device, with exactly the following names, before you run BATCH:

BASIC.SAV	(BASIC users only)
BA.SYS	
BATCH.SAV	
CREF.SAV	(MACRO users only)
SYSLIB.OBJ	(FORTRAN and MACRO users)
FORTRA.SAV	(FORTRAN users only)
LINK.SAV	
MACRO.SAV	(MACRO users only)
PIP.SAV	
DIR.SAV	

A.2 Control Statement Format

For input to RT–11 BATCH, you can generate a file with the RT–11 editor and use any RT–11 input device, or you can use punched cards from the card reader. In both cases, the input consists of BATCH control statements. A BATCH control statement is divided into three fields, separated from one another by spaces: command fields, specification fields, and comment fields. The control statement has the syntax:

$command/option specification/option [!comment]

Each control statement requires a specific combination of command and specification fields and options (see Section A.4). Control statements cannot be longer than 80 characters, excluding multiple spaces, tabs, and comments. You can use a hyphen (-) as a line continuation character to indicate that the control statement is continued on the next line (see Table A–4). Even if you use the line continuation character, the maximum control statement length is still 80 characters.

The following example of a $FORTRAN command illustrates the various fields in a control statement.

```
$FORTRAN/LIST/RUN  PROGA/LIBRARY  PROGB/EXE  !RUN FORTRAN
```

command/options spec fields/options comment field

A.2.1 Command Fields

The command field in a BATCH control statement indicates the operation to be performed. It consists of a command name and certain command field options. Indicate the command field with a $ in the first character position and terminate it with a space, tab, blank, or carriage return.

A.2.1.1 Command Names – The command name must appear first in a BATCH control statement and have a dollar sign ($) in the first position of the command (for example, $JOB). No intervening spaces are allowed in the command name. BATCH recognizes only two forms of a command name: the full name, and an abbreviation consisting of $ and the first three characters of the command name. For example, you can enter the $FORTRAN command as:

```
$FORTRAN
```

or

```
$FOR
```

You cannot enter it as:

```
$FORT
```

or

```
$FORTR
```

A.2.1.2 Command Field Options – Options that appear in a command field are command qualifiers. Their functions apply to the entire control statement. All option names must begin with a slash (/) that immediately follows the command name. Table A–1 describes the command field options for BATCH and indicates the commands on which you can use them. Those option characters that appear in square brackets are optional. The command field options are described in greater detail in the sections dealing with the appropriate commands.

NOTE

All /NO options are the defaults, except the /WAIT option in the $MOUNT and $DISMOUNT commands and the /OBJECT option in the $LINK command.

A.2.2 Specification Fields

Specification fields immediately follow command fields in a BATCH control statement and apply only to the fields they follow. Use them to name the devices and files involved in the command. You must separate these fields from the command field, and from each other, by blanks or spaces.

If a specification field contains more than one file to be used in the same operation, separate the files by a plus (+) sign. For example, to assemble files F1 and F2 to produce an object file F3 and a temporary listing file, type:

```
$MACRO/LIST F1+F2/SOURCE F3/OBJECT
```

Table A–1: Command Field Options

Option	Function
/BAN[NER]	Prints the header of the job on the log file. BATCH allows this option only on the $JOB command. Note that BATCH outputs the $JOB command line to the log device sixty times.
/NOBAN[NER]	Does not print a job header.
/CRE[F]	Produces a cross-reference listing during compilation. BATCH allows this option only on the $MACRO command.
/NOCRE[F]	Does not create a cross-reference listing.
/DEL[ETE]	Deletes input files after the operation completes. BATCH allows this option on the $COPY and $PRINT commands.
/NODEL[ETE]	Does not delete input files after operation completes.
/DOL[LARS]	The data following this command can have a $ in the first character position of a line. BATCH allows this option on the $CREATE, $DATA, $FORTRAN, and $MACRO commands. BATCH terminates reading data when you use one of the following commands or when it encounters a physical end-of-file on the BATCH input stream: $JOB $EOD $SEQUENCE $EOJ
/NODOL[LARS]	The data following this command cannot have a $ in the first character position; a $ in the first character position means a BATCH control command.
/LIB[RARY]	Includes the default library in the link operation. BATCH allows this option on the $LINK and $MACRO commands.
/NOLIB[RARY]	Does not include the default library in the link operation.
/LIS[T]	Produces a temporary listing file (see Section A.2.5) on the listing device (LST:) or writes data images on the log device (LOG:). BATCH allows this option on the $BASIC, $CREATE, $DATA, $FORTRAN, $JOB, and $MACRO commands. When you use /LIST on the $JOB command, /LIST sends data lines in the job stream to the log device (LOG:).
/NOLIS[T]	Does not produce a temporary listing file.
/MAP	Produces a temporary link map on the listing device (LST:). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands.
/NOMAP	Does not create a MAP file.
/OBJ[ECT]	Produces a temporary object file as output from compilation or assembly (see Section A.2.5). BATCH allows this option on the $FORTRAN, $LINK, and $MACRO commands. When you use /OBJECT on $LINK, BATCH includes temporary files in the link operation.

(Continued on next page)

Table A–1: Command Field Options (Cont.)

Option	Function
/NOOBJ[ECT]	Does not produce an object file as output of compilation; with $LINK, does not include temporary files in the link operation.
/RT11	Sets BATCH to operate in RT–11 mode (see Section A.5). BATCH allows this option only on the $JOB command.
/NORT11	Does not set BATCH to operate in RT–11 mode.
/RUN	Links (if necessary) and executes programs compiled since the last link-and-go operation or start of job. BATCH allows this option on the $BASIC, $FORTRAN, $LINK, and $MACRO commands.
/NORUN	Does not execute or link and execute the program after performing the specified command.
/TIM[E]	Writes the time of day to the log file when BATCH executes. BATCH allows this option only on the $JOB command. This command writes the time after each command that begins with a dollar sign ($).
/NOTIM[E]	Does not write the time of day to the log file.
/UNI[QUE]	Checks for unique spelling of options and keynames (see Section A.4.13). BATCH allows this option only on the $JOB command.
/NOUNI[QUE]	Does not check for unique spelling.
/WAI[T]	Pauses for operator action. BATCH allows this option on the $DISMOUNT, $MESSAGE, and $MOUNT commands.
/NOWAI[T]	Does not pause for operator action.
/WRI[TE]	Indicates that the operator is to WRITE-ENABLE a specified device or volume. BATCH allows this option only on the $MOUNT command.
/NOWRI[TE]	Indicates that no writes are allowed or that the specified volume is read-only; informs the operator, who must WRITE-LOCK the appropriate device.

If you need to repeat a command for more than one field specification, separate the files by a comma (,). For example, the following command assembles F1 to produce F2, a temporary listing file, and a map file F3. It then assembles F4 and F5 to produce F6 and a temporary listing file.

```
$MACRO/LIST F1/SOURCE F2/OBJECT F3/MAP,F4+F5/SOURCE-
 F6/OBJECT
```

Depending on the command you use, specification fields can contain a device specification, file specification, or an arbitrary ASCII string. You can use an appropriate specification field option (see Table A–3) with any of these three items.

A.2.2.1 Physical Device Names – Represent each device in an RT–11 BATCH specification field with a standard two- or three-character device name. Table 3–1 in Chapter 3 of the *RT–11 System User's Guide* lists each name and its related device. If you do not specify a unit number for devices that have more than one unit, BATCH assumes unit 0.

In addition to the permanent names shown in Table 3–1, you can assign logical device names to devices. A logical device name takes precedence over a physical name, thus providing device independence. With this feature, you do not need to rewrite a program that is coded to use a specific device if the device is unavailable. For example, DK: is initially assigned to the system device, but you can assign that name to diskette unit 1 (DX1:) with an RT–11 monitor ASSIGN command.

You must assign certain logical names prior to running any BATCH job. BATCH uses these logical names as default devices. These names are:

LOG: BATCH log device (cannot be magtape or cassette)
LST: Default for listing files generated by BATCH stream

The following are not legal device names in RT–11; if you use them, the operator must assign them as logical names with the ASSIGN command. You can use these names in BATCH streams written for other DIGITAL systems.

DF: Fixed-head disk (RF)

LL: Line printer with uppercase and lowercase characters

M7: 7-track magtape

M9: 9-track magtape

PS: Public storage (DK: as assigned by RT–11)

Refer to the ASSIGN keyboard command in the *RT–11 System User's Guide* and Section A.7.1 in this manual for instructions on assigning logical names to devices.

A.2.2.2 File Specifications – You can reference files symbolically in a BATCH control statement with a name of up to six alphanumeric characters followed, optionally, by a period and a file type of three alphanumeric characters. Tabs and embedded spaces are not allowed in either the file name or file type. The file type generally indicates the format of a file. It is good practice to conform to the standard file types for RT–11 BATCH. If you do not specify a file type for an output file, BATCH and most other RT–11 system programs assign appropriate default file types. If you do not specify a file type for an input file, the system searches for that file name with a default file type. Table A–2 lists the standard file types used in RT–11 BATCH.

A.2.2.3 Wildcard Construction – You may use wildcards in certain BATCH control statements (such as, $COPY, $CREATE, $DELETE, $DIRECTORY, $PRINT). You can use the asterisk as a wildcard to designate the entire file name or file type. See Chapter 4 of the *RT–11 System User's Guide* for a complete description of the wildcard construction.

Table A–2: BATCH File Types

File Type	Explanation
.BAS	BASIC source file (BASIC input)
.BAT	BATCH command file
.CTL	BATCH control file generated by the BATCH compiler
.CTT	BATCH temporary file generated by the BATCH compiler
.DAT	BASIC or FORTRAN data file
.DIR	Directory listing file
.FOR	FORTRAN IV source file (FORTRAN input)
.LST	Listing file
.LOG	BATCH log file
.MAC	MACRO source file (MACRO or SRCCOM input)
.MAP	Link map output from $LINK operation
.OBJ	Object file output from compilation or assembly
.SOU	Temporary source file
.SAV	Runnable file or program image output from $LINK

NOTE

You cannot use embedded wild cards (* or %) in BATCH control statements. However, you can use them in the keyboard monitor commands if you use the RT–11 mode of BATCH.

A.2.2.4 Specification Field Options – Specification field options follow file specifications in a BATCH control statement and designate how the file will be used. These options apply only to the field in which they appear. Option names begin with a slash. The specification field options for RT–11 BATCH are listed in Table A–3. Optional characters in the option names are in square brackets.

A.2.3 Comment Fields

Comment fields, which document a BATCH stream, are identified by an exclamation point (!) appearing anywhere except in the first character position of the control statement. BATCH treats any character following the ! and preceding the carriage return/line feed combination as a comment. For example:

```
$RUN PIP      !DELETE FILES ON DK:
```

This command runs the RT–11 system program PIP. BATCH ignores the comment.

Table A–3: Specification Field Options

Option	Explanation
/BAS[IC]	BASIC source file
/EXE[CUTABLE]	Indicates the executable program image file to be created as the result of a link operation
/FOR[TRAN]	FORTRAN source file
/INP[UT]	Input file; default if you specify no options
/LIB[RARY]	Library file to be included in link operation (prior to default library)
/LIS[T]	Listing file
/LOG[ICAL]	Indicates that the device is a logical device name; use in $DISMOUNT and $MOUNT commands
/MAC[RO]	MACRO source file
/MAP	Linker map file
/OBJ[ECT]	Object file (output of assembly or compilation)
/OUT[PUT]	Output file
/PHY[SICAL]	Indicates physical device name
/SOU[RCE]	Indicates source file
/VID	Volume identification

You can also include comments as separate comment lines by typing a $ in character position 1, followed immediately by the ! operator and the comment. For example:

```
$!DELETE FILES ON DK:
```

A.2.4 BATCH Character Set

The RT–11 BATCH character set is limited to the 64 uppercase characters (ASCII 40 through 137). The current ASCII set is assumed (character 137 is underscore and not left-arrow, and character 136 is circumflex, not up-arrow). The BATCH job control language does not support any control characters other than tab, carriage return, and line feed.

Table A–4 shows how BATCH normally interprets certain characters. Character interpretations are different if you use RT–11 mode (see Section A.5).

Table A–4: Character Explanation

Character	Explanation
space	Specification field delimiter. It separates arguments in control statements. BATCH considers any string of consecutive spaces and tabs (except in quoted strings) as a blank (that is, equivalent to a single space).
!	Comment delimiter. The input routine ignores all characters after the exclamation point, up to the carriage return/line feed combination.
"	Passes a text string containing delimiting characters where the normal precedence rules would create the wrong action. For example, use it to include a space in a volume identification (/VID).
$	BATCH control statement recognition character. A dollar sign ($) in the first character position of a BATCH input stream line indicates that the line is a control statement.
.	Delimiter for file type.
–	Indicates line continuation if the character after the hyphen is one of the following: • A carriage return/line feed • Any number of spaces or tabs followed by a carriage return/line feed • A comment delimiter (!) • Spaces followed by a comment delimiter (!) If any other character follows the hyphen, the hyphen is assumed to be a minus sign indicating a negative value in an option
/	Precedes an option name. An alphanumeric string must immediately follow it.
0–9	Numeric string components.
:	Immediately follows a device name. You can also use it to separate an option name from its value or to separate an option value from its subvalue (you can use : interchangeably with = for this purpose).
A–Z	Alphabetic string components.
=	Separates an option name from a value.
\	Illegal character except when it precedes a directive to the BATCH run-time handler from the operator (see Section A.7.3). (To include \ in an RT–11 mode command, use \\.)
+	File delimiter. Separates multiple files in a single specification field. Also indicates a positive value in options.
,	Separates sets of arguments for which the command is to be repeated.
*	A wildcard in utility command file specifications.
CR/LF	Carriage return/line feed. It indicates end-of-line (or end of logical record) for records in the BATCH input stream.

A.2.5 Temporary Files

When you do not include field specifications in a BATCH command line, BATCH sometimes generates temporary files. For example, you can enter a $FORTRAN command that is followed in the BATCH stream by the FORTRAN source program as:

```
$FORTRAN/RUN/OBJECT/LIST
 FORTRAN source program
$EOD
```

This command generates a temporary source file from the source statements that follow, a temporary object file, a temporary listing file, and a temporary memory image file.

BATCH sends temporary files to the default device (DK:) or the listing device (LST:) according to their type. If the device is file-structured, BATCH assigns file names and file types as follows:

nnnmmm.LST	for temporary listing files (sent to LST:)
nnnmmm.MAP	for temporary map files (sent to LST:)
nnnppp.OBJ	for temporary object files (sent to DK:)
000000.SAV	for temporary memory image files (sent to DK:)
nnnppp.SOU	for temporary source files (sent to DK:)

where:

nnn — represents the last three digits of the sequence number assigned to the job by the $SEQUENCE command (see Section A.4.22). Thus, a sequence number of 12345 produces a file name beginning 345. If you do not use the $SEQUENCE command, BATCH sets nnn to 000.

mmm — represents the number of listing (or map) files BATCH generated since the BATCH run-time handler (BA.SYS) was loaded. The first such file, listing or map, is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. Thus, the second listing file produced under job sequence number 12345 is 345001.LST, and the first map file produced is 345000.MAP.

ppp — represents the number of object or source files in the current BATCH run. The first such file (object or source) is 000. Each time BATCH generates a new temporary file, it increments the file name by 1. BATCH resets these file names to 000 every time you run BATCH and after every $LINK, $MACRO, or $FORTRAN command that uses the temporary files.

A.3 General Rules and Conventions

You must adhere to the following general rules and conventions associated with RT–11 BATCH processing.

1. Always place a dollar sign ($) in the first character position of a command line.
2. Each job must have a $JOB and $EOJ command (or card).
3. You can spell out command and option names entirely or you can specify only the first three characters of the command and required characters of the option.
4. Specify wildcard construction (*) only for the utility commands ($COPY, $CREATE, $DELETE, $DIRECTORY, and $PRINT) and for commands that normally accept wildcards in RT–11 mode.
5. Include comments at the end of command lines or in a separate comment line. When you include comments in a command line, place them after the command but precede them by an exclamation mark.
6. Include only 80 characters per control statement (card record), excluding multiple spaces, tabs, and comments.
7. When you omit file specifications from BATCH commands and supply data in the BATCH stream, the system creates a temporary file with a default name (see Section A.2.5).
8. You can use the RT–11 monitor type-ahead feature only with BATCH handler directives (see Section A.7.3) to be inserted into a BATCH program. No other terminal input (except input to a foreground program) can be entered while a BATCH stream is executing.
9. You cannot use an indirect command file to call BATCH.

A.4 Commands

Place BATCH commands in the input stream to indicate to the system which functions to perform in the job. All BATCH commands have a dollar sign ($) in the first character position (for example, $JOB). Intervening spaces are not allowed in command names. The command name must always start in the first character position of the line (card column 1).

BATCH commands are presented in alphabetical order in this chapter for ease of reference. However, if you are not familiar with BATCH, read the commands in a functional order as listed in Table A–5. The characters shown in square brackets are optional.

Table A–5: BATCH Commands

Command	Section	Function
$SEQ[UENCE]	A.4.22	Assigns an arbitrary identification number to a job.
$JOB	A.4.13	Indicates the start of a job.
$EOJ	A.4.11	Indicates the end of a job.
$MOU[NT]	A.4.18	Signals the operator to mount a volume on a device and optionally assigns a logical device name.
$DIS[MOUNT]	A.4.9	Signals the operator to dismount a volume from a device and deassigns a logical device name.
$FOR[TRAN]	A.4.12	Compiles a FORTRAN source program.
$BAS[IC]	A.4.1	Compiles a BASIC source program.
$MAC[RO]	A.4.16	Assembles a MACRO source program.
$LIB[RARY]	A.4.14	Specifies libraries for BATCH to use in link operations.
$LIN[K]	A.4.15	Links modules for execution.
$RUN	A.4.21	Causes a program to execute.
$CAL[L]	A.4.2	Transfers control to another BATCH file, executes that BATCH file, and returns to the calling BATCH stream.
$CHA[IN]	A.4.3	Passes control to another BATCH file.
$DAT[A]	A.4.6	Indicates the start of data.
$EOD	A.4.10	Indicates the end of data.
$MES[SAGE]	A.4.17	Issues a message to the operator.
$COP[Y]	A.4.4	Copies files.
$CRE[ATE]	A.4.5	Creates new files from data included in the BATCH stream.
$DEL[ETE]	A.4.7	Deletes files.
$DIR[ECTORY]	A.4.8	Provides a directory of the specified device.
$PRI[NT]	A.4.19	Prints files.
$RT[11]	A.4.20	Specifies that the following lines are RT–11 mode commands.

For each command listed below, the term filespec represents a device name, or file name, and a file type. Filespec has this form:

dev:filnam.typ

As a general rule, BATCH assumes device DK: if you omit a device specification.

A.4.1 $BASIC

The $BASIC command calls RT–11 single-user BASIC to execute a BASIC source program. The $BASIC command has the following syntax:

$BASIC[/option...] [filespec/option]] [!comments]

where:

/option — indicates an option you can append to the $BASIC command. The options are as follows:

- /RUN — indicates that BATCH should execute the source program.
- /NORUN — indicates that BATCH should only compile the program and send error messages to the log file.
- /LIST — writes data images that are contained in the job stream to the log file (LOG:).
- /NOLIST — writes data images to the log file only if you specify $JOB/LIST.

filespec — indicates the name and type of the source file and the device on which it resides. If you omit the file type, BATCH assumes .BAS. If you omit this specification, the source statements must immediately follow the $BASIC command in the input stream.

Terminate the source program after a $BASIC statement with either a $EOD command or with any other BATCH command that starts with a $ in the first position.

/option — indicates an option that can follow the source file name. BATCH assumes any file name with no option appended is the name of a source file. This option can have one of the following values (or you can omit it):

- /BASIC — indicates that the file name you specify is a BASIC source program.
- /SOURCE — performs the same function as /BASIC.
- /INPUT — performs the same function as /BASIC.

You can follow the $BASIC command with the source program, BASIC commands (such as RUN), or data. The following two BATCH streams, for example, produce the same results (but BATCH does not echo the same output format for both streams).

```
$BASIC
10 INPUT A
20 PRINT A
30 END
RUN
123
$EOD
```

```
$BASIC/RUN
10 INPUT A
20 PRINT A
30 END
$DATA
123
$EOD
```

A.4.2 $CALL

The $CALL command transfers control to another BATCH control file, temporarily suspending execution of the current control file. BATCH executes the called file until it reaches $EOJ or until the job aborts; control then returns to the statement following the $CALL in the originating BATCH control file. You can nest calls up to 31 levels. BATCH includes the log file for the called file in the log file for the originating BATCH program. (See NOTE following the $EOJ command.)

The syntax of the $CALL command is:

$CALL filespec[!comments]

Options are not allowed in the $CALL command. BATCH saves $JOB command options across a $CALL; however, they do not apply to the called BATCH file. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the called BATCH stream before execution.

NOTE

If the called program generates temporary files, those files can supersede existing temporary files if the two jobs have the same sequence number. For example, consider the following two BATCH streams:

```
$FOR/OBJ A      $FOR/OBJ A
$FOR/OBJ B      $CALL C
$LINK/RUN       $FOR/OBJ B
```

The called BATCH file (C.BAT) contains the following:

```
$JOB
$FOR/OBJ A1
$FOR/OBJ B1
$LINK/RUN
$EOJ
```

The temporary object files C.BAT generates change the behavior of the previous two BATCH statement sequences. The first temporary file created by C.BAT (000000.OBJ) supersedes the temporary file produced by the first $FORTRAN command (000000. OBJ). You can avoid this situation by giving the BATCH job C.BAT a unique sequence number (see Section A.4.22).

A.4.3 $CHAIN

The $CHAIN command transfers control to a named BATCH control file but does not return to the input stream that executed the $CHAIN command. The syntax of the $CHAIN command is:

$CHAIN filespec[!comments]

BATCH does not permit options in the $CHAIN command. If you specify .CTL as the file type, BATCH assumes a precompiled BATCH control file. If you do not specify a file type, BATCH assumes .BAT and compiles the chained BATCH stream before execution.

A $EOJ command should always follow the $CHAIN command in the BATCH stream.

NOTE

> The values of BATCH run-time variables remain constant across a $CALL, $CHAIN, or return from call. See Section A.5.2.2 for a description of these variables.

Use the $CHAIN command to transfer control to programs that you need to run only once at the end of a BATCH stream. For example, you could use the following BATCH program (PRINT.BAT) to print and then delete all temporary listing files generated during the current BATCH job.

```
$JOB                          !PRINT ALL LIST FILES
$PRINT/DELETE *.LST
$EOJ
```

You could then run PRINT.BAT with the $CHAIN command as follows:

```
$JOB
$MACRO/RUN                     A ALST/LIST
$MACRO/RUN                     B BLST/LIST
$CHAIN PRINT
$EOJ
```

A.4.4 $COPY

The $COPY command copies files in image mode from one device to another. You can use the wild card construction (see Section A.2.2.3) in the input and output file specifications. You can concatenate several input files to form one output file (as long as the output specification does not contain a wild card). The $COPY command has the following syntax:

$COPY[/option] output-filespec[...,output-filespec]/OUTPUT-input-filespec[...,input-filespec][/INPUT][!comments]

where:

/option — indicates options that you can append to the $COPY command.

- /DELETE — deletes input files after the copy operation.
- /NODELETE — does not delete input files after the copy operation.

output-filespec represents an output file; you must specify a file type.

/OUTPUT	indicates that a file specification is for an output file.
input-filespec	represents a file to be copied. (BATCH copies files to the output file in the order that you list them, except when you use wildcards.)
/INPUT	indicates that a file specification is for an input file; if you do not specify an option, BATCH assumes INPUT.

The following are examples of the $COPY command:

```
$COPY *.BAS/OUTPUT DT1:*.BAS
```

This command copies all files with the file type .BAS from the DECtape on unit 1 to the default storage device DK:.

```
$COPY FILE2.FOR/OUTPUT FILE0.FOR+FILE1.FOR
```

This command merges the input files FILE0.FOR and FILE1.FOR to form one file called FILE2.FOR and stores FILE2.FOR on device DK:.

```
$COPY *.*/OUT DT0:*.FOR, DT1:*.*/OUT DT0:*.*
```

This command copies all files with the file type .FOR from DT0: to DK: and all files on DT0: to DT1:.

A.4.5 $CREATE

The $CREATE command generates a file from data records that follow the $CREATE command in the input stream. An error occurs if the data does not immediately follow the $CREATE command. You cannot precede the data records with a $DATA command.

You can follow the $CREATE data with a $EOD command to signify the end of data, or you can use any other BATCH control statement to indicate end of data and initiate a new function. The $CREATE command has the following syntax:

$CREATE[/option...] filespec [!comments]

where:

/option	indicates an option you can append to the $CREATE command. The options are:	
	/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
	/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
	/LIST	writes data image lines to the log file.

/NOLIST	does not write data image lines to the log file. If you specify $JOB/LIST, BATCH ignores this option.
filespec	represents the file you want to create.

NOTE

If you use the /DOLLARS option, you must follow the last data record with a $EOD command (see Table A–1).

The following is an example of the $CREATE command:

```
$CREATE/LIST PROG.FOR
FORTRAN source file
$EOD
```

The data records following the $CREATE command become a new file (PROG.FOR) on the default device (DK:). BATCH generates a listing on logical device LOG:.

A.4.6 $DATA

Use the $DATA command to include data records in the input stream. Data you include in this manner needs no file name. BATCH transfers the data to the appropriate program as though it were input from the console terminal. For example, you can follow the $RUN command for a particular program by a $DATA command and the data records for the program to process. The data records must be valid data for the program that is to use them.

The $DATA command has the following syntax:

$DATA[/option...] [!comments]

Four options that you can use with the $DATA command are as follows:

/DOLLARS	Indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	Indicates that a $ cannot be in the first character position of a line.
/LIST	Writes data image lines to the log file.
/NOLIST	Does not write data images to the log file. If you specify $JOB/LIST, BATCH ignores this option.

NOTE

Any command beginning with a $ normally follows the last data record. However, if you specify $DATA/DOLLARS, you must follow the last data record with $EOD.

The following example shows data entered into a BASIC program (TEST1.BAS).

```
$BASIC/RUN TEST1.BAS
$DATA
25,75,125,146
180,210,520,874
$EOD
```

A.4.6.1 Using $DATA with FORTRAN Programs — When you use the $DATA command to provide input to a FORTRAN program, you must insert a CTRL/Z into the BATCH file after the last data line and before $EOD (or before the next BATCH command if you do not use $EOD). This procedure permits FORTRAN to properly detect an end-of-file after it reads the last data line. For example:

```
$FORTRAN/RUN  A.FOR
$DATA
1
2
3
^Z (RET) (LF)
$EOD
$RUN PIP
```

The above program reads three numbers from the input stream and then detects an end-of-file when it attempts to read a fourth number. If you include an END=n statement in your FORTRAN program, statement n gets control when the end-of-file is detected. If the CTRL/Z <RET> <LF> is not present, the program aborts when it reaches $EOD and never executes the END=n statement.

A.4.7 $DELETE

Use the $DELETE command to delete files from the device you specify. This command has the syntax:

$DELETE filespec[...,filespec][!comments]

where:

filespec represents the name of a file to be deleted

The following example deletes all files named TEST1 on the default device DK:.

```
$DELETE TEST1.*
```

The following example deletes all files with .FOR file types on DT1:, then deletes all files with .MAC file types on DK:.

```
$DELETE DT1:*.FOR,*.MAC
```

A.4.8 $DIRECTORY

The $DIRECTORY command outputs a directory of the device you specify to a listing file. If you do not specify a listing file, the listing goes to the BATCH log file. This command has the syntax:

$DIRECTORY [filespec/LIST] [filespec[...,filespec]][/INPUT] [!comments]

where:

filespec/LIST	indicates the name of the directory listing file
filespec/INPUT	indicates the input files to be included in the directory (default)

The following command outputs a directory of the device DK: to the BATCH log file.

```
$DIRECTORY
```

This next command creates on the device DK: a directory file (FOR.DIR) that contains the names, lengths, and dates of creation of all FORTRAN source files on that device.

```
$DIRECTORY FOR.DIR/LIST *.FOR
```

A.4.9 $DISMOUNT

The $DISMOUNT command removes the logical device name assigned by a $MOUNT command. When BATCH encounters $DISMOUNT while executing a job, it prints the entire $DISMOUNT command line on the console terminal. This message tells the operator which device to unload. This command has the syntax:

$DISMOUNT[/option] logical-device-name:[/LOGICAL] [!comments]

where:

/option	indicates an option you can append to the $DISMOUNT command. The options are:
	/WAIT — indicates that the job must pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell at the terminal, prints the physical device name to be dismounted followed by a question mark (?), and waits for a response. (At this point you can enter input to the BATCH handler. See Section A.7.3.)

/NOWAIT — does not pause for operator response; BATCH prints the physical device name to be dismounted.

logical-device-name: is the logical device name to be deassigned from the physical device.

/LOGICAL — identifies the device specification as a logical device name.

The following example instructs the operator to dismount the physical device with the logical device name OUT: and removes the logical assignment of device OUT:. In this example, OUT: is DT0:. The operator dismounts DT0: and then types a carriage return.

```
$DISMOUNT/WAIT OUT:/LOGICAL
DT0?
```

A.4.10 $EOD

The $EOD command indicates the end-of-data record or the end of a source program in the job stream. The syntax of this command is:

$EOD [!comments]

The $EOD command can signal the end of data associated with any of the following commands:

$BASIC	$FORTRAN
$CREATE	$MACRO
$DATA	

In the following example, the $EOD command indicates the end of a source program that is to be compiled, linked, and executed.

```
$FORTRAN/RUN
source program
$EOD
```

A.4.11 $EOJ

The $EOJ command indicates the end of a job. This command must be the last statement in every BATCH job. The command has the following syntax:

$EOJ [!comments]

If BATCH encounters a $JOB command, a $SEQUENCE command, or a physical end-of-file in the input stream before $EOJ, an error message appears in the log file.

NOTE

Make sure that the $EOJ command is the last line in a BATCH file.

A.4.12 $FORTRAN

The $FORTRAN command calls the FORTRAN compiler to compile a source program. Optionally, this command can provide printed listings or list files and can produce a link map in the listing. The $FORTRAN command has the following syntax:

```
$FORTRAN[/option...] [source-filespec[/option]] [filespec/OBJECT]-
[filespec/LIST] [filespec/EXECUTE]-
[filespec/MAP] [filespec/LIBRARY] [!comments]
```

where:

/option — indicates an option you can append to the $FORTRAN command. The options are as follows:

/RUN	indicates that FORTRAN is to compile the source program, link it with the default library, and execute it. The default library is SYSLIB.OBJ. You can change it with the $LIBRARY command.
/NORUN	compiles the program only.
/OBJECT	produces a temporary object file.
/NOOBJECT	does not produce a temporary object file.
/LIST	produces a list file on the listing device (LST:).
/NOLIST	does not produce a list file.
/MAP	produces a link map on the listing device (LST:).
/NOMAP	does not create a MAP file.
/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ cannot be in the first character position of a line.

source-filespec — indicates the device, file name, and file type of the FORTRAN source file. If you do not specify the file name, the $FORTRAN source statements must immediately follow the $FORTRAN command in the input stream; BATCH generates a temporary source file that it deletes after FORTRAN compiles the temporary source file (see Section A.2.5).

You can terminate the source program included after a $FORTRAN statement by either a $EOD command or by any other BATCH command. If, however, you use dollar signs in the first position in the source program, you must enter the source program with $CREATE/DOLLARS. In this case, you cannot use $FORTRAN/DOLLARS.

/option — represents an option that can have one of the following values:

- /FORTRAN — indicates that the file name you specify is a FORTRAN source program. BATCH assumes that any file name with no option appended is the name of a source file.
- /SOURCE — performs the same function as /FORTRAN.
- /INPUT — performs the same function as /FORTRAN.

filespec/OBJECT — indicates the device, file name, and file type of the object file produced by compilation. The object file remains on the device you specify after the job finishes. You must follow the object file specification, if you include it, with the /OBJECT option.

If you omit the object file specification but specify $FORTRAN/OBJECT, BATCH creates a temporary object file. BATCH includes this temporary file in any $LINK operations that follow it in the job, and deletes it after the link operation.

filespec/LIST — indicates the name you assign to the list file created by the compiler. BATCH does not automatically print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. Follow the list file specification with the /LIST option.

filespec/EXECUTE — indicates the name you assign to a memory image file. Follow the memory image file specification with the /EXECUTE option. If you do not include this field, BATCH generates a temporary memory image file (see Section A.2.5) and then deletes the temporary file.

filespec/MAP — indicates the name you assign to the link map file created by the linker. Follow the map specification with the /MAP option.

filespec/LIBRARY indicates that BATCH must include the file you specify in the link procedure as a library before SYSLIB.OBJ. The file must be a library file (produced by the RT–11 librarian). Follow the library specification with the /LIBRARY option.

The following command calls FORTRAN to compile and execute a source program named PROGA.FOR.

```
$FORTRAN/RUN PROGA.FOR
```

The next command sequence compiles the FORTRAN program but does not produce an object file. BATCH creates a temporary listing file on LST:.

```
$FORTRAN/NOOBJ/LIST
source program
$EOD
```

NOTE

See Section A.4.6.1 for instructions on using the $DATA command with FORTRAN programs.

A.4.13 $JOB

The $JOB command indicates the beginning of a job. Each job must have its own $JOB command. This command has the following syntax:

$JOB[/option...] [!comments]

BATCH allows the following options in the $JOB command:

/BANNER	Prints a header (a repetition of the $JOB line or card) on the log file.
/NOBANNER	Does not print a job header.
/LIST	Writes data image lines that are contained in the job stream to the log file.
/NOLIST	Writes data image lines to the log file only when a /LIST option exists on a $BASIC, $CREATE, or $DATA command that has data lines following it.
/RT11	If no $ appears in column 1 when BATCH expects one, BATCH assumes that the line or card is an RT–11 mode command (see Section A.5).
/NORT11	Does not process RT–11 mode commands.
/TIME	Writes the time of day to the log file when BATCH executes command lines (except $DATA command lines).

/NOTIME Does not write the time of day.

/UNIQUE Checks for unique spelling of options and keynames. When you use this option, you can abbreviate commands and options to the fewest number of characters that still make their names unique. For example, you can abbreviate the /DOLLARS option to /DO since no other option begins with the characters DO.

/NOUNIQUE Checks only for normal option and keyname spellings.

End each job with a $EOJ command if you want to run it. If an input stream consists of more than one job, BATCH automatically terminates one job when it encounters the $JOB command for the next job. BATCH will never run a job terminated with another $JOB command; instead, an error message will appear in the log.

The following $JOB command writes the time of day to the log file before BATCH executes each command beginning with a $. It also accepts unique abbreviations of BATCH commands and options.

```
$JOB/TIME/UNIQUE
```

A.4.14 $LIBRARY

The $LIBRARY command lets you specify a list of library files for inclusion in FORTRAN links or other link operations that have the /LIBRARY option. By default, the list of libraries contains only SYSLIB.OBJ, the RT–11 system library. This command has the syntax:

$LIBRARY filespec [!comments]

or

$LIBRARY filespec + SYSLIB [!comments]

where:

filespec represents a library file; the default file type is .OBJ

SYSLIB is the RT–11 system library that you create at system generation

Libraries are linked in order of their appearance in the $LIBRARY command.

The following example shows two libraries (LIB1.OBJ and LIB2.OBJ) that are included in FORTRAN links before SYSLIB.OBJ.

```
$LIBRARY LIB1.OBJ+LIB2.OBJ+SYSLIB.OBJ
```

A.4.15 $LINK

Use the $LINK command to produce memory image files from object files. This command links any files you may specify with any temporary object files created since the last link or link-and-go operation.

Temporary object files are those files you create as a result of a $FORTRAN or $MACRO command without naming an object file (with the /OBJECT option) by suppressing an object file (with the /NOOBJECT option). Create permanent object files by using the /OBJECT option on a $FORTRAN or $MACRO file descriptor.

BATCH links files in the following order:

1. Temporary files — in the order in which they were compiled
2. Permanent files — in the order in which they are specified in the $LINK command
3. Any library specified by the $LINK command — provided that unresolved references remain
4. The default library list — if you specified $LINK/LIBRARY

The syntax for this command is:

```
$LINK[/option...] [filespec/OBJECT] [filespec/LIBRARY]-
[filespec/MAP] [filespec/EXECUTE] [!comments]
```

where:

/option — indicates an option that you can append to the $LINK command. The options are as follows:

- /LIBRARY — includes the RT–11 system library (SYSLIB.OBJ) and any default libraries specified in the $LIBRARY command in this $LINK operation. Use this option when the files being linked do not include any temporary FORTRAN object files. You can also use it when you specify $FORTRAN without the /RUN or /MAP option, but want to search the default library list for unresolved references.
- /NOLIBRARY — does not include the default libraries.
- /MAP — produces a temporary load map on the listing device (LST:).

/NOMAP	does not produce a map file.
/OBJECT	includes temporary object files in the link. If you specify neither /OBJECT nor /NOOBJECT, BATCH assumes $LINK /OBJECT.
/NOOBJECT	does not include temporary files in the link.
/RUN	executes the memory image files associated with this $LINK command when the link is complete.
/NORUN	only links the program and does not execute it.

filespec/OBJECT	indicates the name of the object file BATCH must link; if you do not specify /OBJECT, BATCH assumes it as the default.
filespec/LIBRARY	indicates that the file you specify is to be included in the link procedure as a library; the file you specify must be a library file (produced by the RT–11 librarian).
filespec/MAP	indicates the load map file BATCH must create as a result of the $LINK command.
filespec/EXECUTE	indicates the memory image file BATCH must create as a result of the $LINK command.

The following command links all temporary object files created since the last $LINK command, or the last $FORTRAN/OBJ or $MACRO/OBJ command.

```
$LINK/RUN
```

The next command links the temporary files and the object files PROG1.OBJ and PROG2.OBJ to form a memory image file named PROGA.SAV. It also creates and outputs a temporary map file.

```
$LINK/MAP PROG1.OBJ+PROG2.OBJ/OBJ PROGA.SAV/EXE
```

A.4.16 $MACRO

The $MACRO command calls the MACRO assembler to assemble a source program and, optionally, to provide printed listings or list files. You must specify any MACRO listing directives in the source program; you cannot enter them at BATCH command level.

The $MACRO command has the following syntax:

$MACRO[/option...] [source-filespec[/option]] [filespec/OBJECT]-
[filespec/LIST] [filespec/MAP] [filespec/LIBRARY]-
[filespec/EXECUTE] [!comments]

where:

/option — indicates an option you can append to the $MACRO command. The options are as follows:

/RUN	assembles, links, and runs the source program.
/NORUN	only assembles the source program.
/OBJECT	produces a temporary object file.
/NOOBJECT	does not produce a temporary object file.
/LIST	produces a listing file on the listing device (LST:).
/NOLIST	does not produce a list file.
/CREF	produces a cross-reference listing during assembly.
/NOCREF	does not produce a cross-reference listing during assembly.
/MAP	produces a link map as part of the listing file on LST:.
/NOMAP	does not create a MAP file.
/DOLLARS	indicates that the data following this command can have a $ in the first character position of a line.
/NODOLLARS	indicates that a $ cannot be in the first character position of a line.
/LIBRARY	includes the default library in the link operation.
/NOLIBRARY	does not include the default library in the link operation.

source-filespec — indicates the name of the source file. If you do not specify a file name, the $MACRO source statements must immediately follow the $MACRO command in the input stream.

You can terminate the source program you include after a $MACRO statement with either a $EOD command or any other BATCH command.

If, however, you include dollar signs in the first position in the source program, use the $CREATE/DOLLARS command to enter the source program. In this case, you cannot use $MACRO/DOLLARS.

/option — can have one of the following values:

- /MACRO — indicates that the file name you specify is a MACRO source program. BATCH assumes that any file name with no option appended is the name of a source file.
- /SOURCE — performs the same function as /MACRO.
- /INPUT — performs the same function as /MACRO.

filespec/OBJECT — indicates the name you assign to the object file produced by compilation. The object file remains on the device you specify after the job finishes. If you include an object file specification, follow it with the /OBJECT option.

If you omit the object file specification but specify $MACRO/OBJECT, BATCH creates a temporary object file. BATCH also includes the temporary object file in any $LINK operations that follow the $MACRO command in the job, and deletes it after the link operation (see Section A.2.5).

filespec/LIST — indicates the name you assign to the list file created by the assembler. BATCH does not print the list file if you assign LST: to a file-structured device, but you can list it using the $PRINT command. The /LIST option must follow the list file specification.

filespec/MAP — indicates the file to which BATCH must output the storage map.

filespec/LIBRARY — indicates that BATCH must include the file you specify in the link procedure as a library. The /LIBRARY option must follow the library file specification.

filespec/EXECUTE — indicates the name you assign to a memory image file. The /EXECUTE option must follow the memory image file specification. If you do not include this field but do use $MACRO/RUN, BATCH generates and runs a temporary memory image file (see Section A.2.5).

The following $MACRO command assembles a program named PROG0.MAC, and creates a temporary object file and a temporary listing file.

```
$MACRO/LIST/OBJECT PROG0.MAC
```

A.4.17 $MESSAGE

Use the $MESSAGE command to issue a message to the operator at the console terminal. It provides a means for the job to communicate with the operator. The $MESSAGE command has the syntax:

$MESSAGE[/option] message [!comments]

where:

/option	indicates an option you can append to the $MESSAGE command. The options are:
	/WAIT — indicates that the job is to pause until the operator either types a carriage return to continue or enters commands to the BATCH handler followed by a carriage return (see Section A.7.3).
	/NOWAIT — does not pause for operator response.
message	is a string of characters that must fit on one console line. BATCH prints the message on the console.

For example, if you include the following message in the input stream:

```
$MESSAGE/WAIT MOUNT SCRATCH TAPE ON MT0:
```

The message:

```
MOUNT SCRATCH TAPE ON MT0:
?
```

appears on the console terminal and a bell sounds. The operator mounts the tape and types carriage return to allow further processing of the job. (See Section A.7.3 for operator interaction with BATCH.)

NOTE

BATCH compresses multiple spaces and tabs in BATCH command lines; therefore, attempts to format $MESSAGE output with tabs or spaces may not provide you with the desired results.

A.4.18 $MOUNT

The $MOUNT command assigns a logical device name and other characteristics to a physical device. When BATCH encounters $MOUNT during the execution of a job, it prints the entire $MOUNT command line on the console terminal to notify the operator which volume to use.

The $MOUNT command has the syntax:

```
$MOUNT[/option...] physical-device-name:[/PHYSICAL][/VID=x]
[logical-device-name:/LOGICAL] [!comments]
```

where:

/option — indicates an option you can append to the $MOUNT command. The options are:

- /WAIT — indicates that the job is to pause until the operator enters a response. If you do not specify either /WAIT or /NOWAIT, BATCH assumes /WAIT. BATCH rings a bell, prints the physical device name and a question mark (?), and waits for a response. (The response can consist of input for the BATCH handler; see Section A.7.3.)
- /NOWAIT — does not pause for operator response. BATCH prints the name of the physical device to be mounted.
- /WRITE — tells the operator to write-enable the volume.
- /NOWRITE — tells the operator to write-protect the volume.

physical-device-name — is required and specifies the physical device name and an optional unit number followed by a colon (for example, DT1:). If you specify a device name without a unit number, the operator can enter one in response to the question mark printed by the $MOUNT command. If you want the operator to supply a unit number, do not use the /NOWAIT option, because it assumes unit 0.

/PHYSICAL — identifies the device specification as a physical unit specification. If you do not specify either /PHYSICAL or /LOGICAL, BATCH assumes /PHYSICAL.

/VID = x
/VID = "x"

provides volume identification. The volume identification is the name physically attached to the volume. Include it to help the operator locate the volume. Use this option only on the physical device file speciication. If x contains spaces, specify it as "x".

NOTE

This volume identification is only a visual check for the operator. Make the identification match the visual label on the volume, not the identification that you wrote onto the volume at initialization time with the INIT/VOLUMEID command.

logical-device-name/LOGICAL

is required to identify any logical device name you may assign to the device. The /LOGICAL option must follow the logical device name specification.

The following command instructs the operator to select a DECtape unit and mount DECtape volume BAT01 on that unit, write-enabled. It informs the operator by printing:

```
$MOUNT/WAIT/WRITE DT:/VID=BAT01 2:/LOGICAL
DT?
```

The operator selects a unit, mounts DECtape volume BAT01, write-enabled, and responds to the question mark by typing the unit number (such as, 1) followed by a carriage return. BATCH assigns logical device name 2 to the physical device (in this case, DT1:) and proceeds.

If no unit number response is necessary, as this command shows,

```
$MOUNT/WAIT/WRITE DT1: 2:/LOGICAL
```

the operator responds with a carriage return after mounting the DECtape and write-enabling the device.

A.4.19 $PRINT

Use the $PRINT command to print the contents of the files you specify on the listing device (LST:). This command has the syntax:

$PRINT[/option] filespec [...,filespec][/INPUT] [!comments]

where:

/option

indicates an option you can append to the $PRINT command. The options are:

/DELETE deletes input files after printing.

/NODELETE does not delete input files after printing.

filespec represents a file to be printed.

/INPUT indicates that the file is an input file; BATCH assumes /INPUT if you omit it.

The following command prints a listing of files with file type .MAC that are stored on default device DK:.

```
$PRINT *.MAC
```

The following example creates listing files for the programs A and B, prints the listing files, and then deletes them.

```
$MACRO A.MAC A/LIST
$MACRO B.MAC B/LIST
$PRINT/DELETE A.LST,B.LST
```

A.4.20 $RT11

The $RT11 command allows the BATCH job to communicate directly with the RT–11 system. DIGITAL recommends that you use RT–11 mode if you use BATCH. This command puts BATCH in RT–11 mode until BATCH encounters a line beginning with $. In RT–11 mode, BATCH interprets all data images as commands to the RT–11 monitor, to RT–11 system programs, or to the BATCH run-time system. The $RT11 command has the syntax:

$RT11 [!comments]

See Section A.5 for a complete description of the RT–11 mode.

A.4.21 $RUN

The $RUN command executes a program for which a memory image file (.SAV) was previously created. It can also run RT–11 system programs.

The $RUN command has the syntax:

$RUN filespec [!comments]

where:

filespec represents the file to be executed. If you omit the file type, BATCH assumes .SAV.

For example, if DIR is on DK:, you can run DIR to print a directory listing:

```
$RUN DIR
$DATA
LP:=DK:/L
$EOD
```

A.4.22 $SEQUENCE

The $SEQUENCE command is an optional command. If you use it, it must immediately precede a $JOB command. The $SEQUENCE command assigns a job an arbitrary identification number. BATCH assigns the last three characters of a sequence number as the first three characters of a temporary listing or object file (see Section A.2.5). If a sequence number is less than three characters long, BATCH fills it with zeroes on the left.

The syntax of this command is:

$SEQUENCE id [!comments]

where:

id	represents an unsigned decimal number that indicates the identification number of a job

The following are examples of the $SEQUENCE command:

```
$SEQUENCE 3            !SEQUENCE NUMBER IS 003
$JOB

$SEQUENCE 100          !SEQUENCE NUMBER IS 100
$JOB
```

A.4.23 Sample BATCH Stream

The following sample BATCH stream creates a MACRO program, assembles and links that program, and runs the memory image file. It then deletes the object, memory image, and source files it created and prints a directory of DK: showing the files the BATCH stream created.

```
$JOB
$MESSAGE          THIS IS AN EXAMPLE BATCH STREAM
$MESSAGE          NOW CREATE A MACRO PROGRAM
$CREATE/LIST      EXAMPL.MAC
.TITLE   EXAMPL FOR BATCH
         .MCALL   .PRINT,.EXIT
START:   .PRINT   #MESSAG
         .EXIT
MESSAG:  .ASCIZ   /EXAMPLE MACRO PROGRAM FOR BATCH/
         .END     START
$EOD
$MACRO   EXAMPL EXAMPL/OBJECT EXAMPL/LIST    !ASSEMBLE
$LINK    EXAMPL EXAMPL/EXECUTE               !AND LINK
$PRINT/DELETE EXAMPL.LST
$MESSAGE          RUN THE MACRO PROGRAM
$RUN     EXAMPL                              !AND EXECUTE
$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC
$MESSAGE          PRINT A DIRECTORY
$DIRECTORY        DK:EXAMPL.*
$MESSAGE          END OF THE EXAMPLE BATCH STREAM
$EOJ
```

To run this batch stream, type the following commands at the console. BATCH prints the messages.

```
. LOAD BA,LP
. ASSIGN LP:LOG
. ASSIGN LP:LST
. R BATCH
* EXAMPL
 THIS IS AN EXAMPLE BATCH STREAM
 NOW CREATE A MACRO PROGRAM
 RUN THE MACRO PROGRAM
 PRINT A DIRECTORY
 END OF THE EXAMPLE BATCH STREAM

END BATCH
.
```

The above sample BATCH stream produces the following log file on the line printer:

NOTE

The amount of free memory and the directory format are variable.

```
$JOB

$MESSAGE        THIS IS AN EXAMPLE BATCH STREAM

$MESSAGE        NOW CREATE A MACRO PROG.

$CREATE/LIST    EXAMPL.MAC

.TITLE  EXAMPLE FOR BATCH
        .MCALL  .PRINT,.EXIT
START:  .PRINT  #MESSAG
        .EXIT
MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        .EVEN
        .END    START

$EOD

$MACRO  EXAMPL EXAMPL/OBJECT EXAMPL/LIST   !ASSEMBLE

ERRORS DETECTED: 0

EXAMPLE FOR BATCH       MACRO V03.00 21-JUN-77 00:05:29 PAGE 1

      1                                 .TITLE  EXAMPLE FOR BATCH
      2                                         .MCALL  .PRINT,.EXIT
      3 000000                          START:  .PRINT  #MESSAG
      4 000006                                  .EXIT
      5 000010      105     130     101 MESSAG: .ASCIZ  /EXAMPLE MACRO PROGRAM FOR BATCH/
        000013      115     120     114
        000016      105     040     115
        000021      101     103     122
        000024      117     040     120
        000027      122     117     107
        000032      122     101     115
        000035      040     106     117
        000040      122     040     102
        000043      101     124     103
        000046      110     000
      6                                         .EVEN
      7 000000'                                 .END    START
```

```
EXAMPLE FOR BATCH        MACRO V03.00 21-JUN-77 00:05:29 PAGE 1-1
SYMBOL TABLE

MESSAG  000010R          START   000000R

. ABS.  000000      000
        000050      001
ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 508 WORDS  ( 2 PAGES)
DYNAMIC MEMORY AVAILABLE FOR  48 PAGES
EXAMPL,EXAMPL=EXAMPL

$LINK   EXAMPL EXAMPL/EXECUTE              !AND LINK

$PRINT/DELETE EXAMPL.LST


$MESSAGE        RUN THE MACRO PROGRAM

$RUN    EXAMPL                                  !AND EXECUTE

EXAMPLE MACRO PROGRAM FOR BATCH

$DELETE EXAMPL.OBJ+EXAMPL.SAV+EXAMPL.MAC


$MESSAGE        PRINT A DIRECTORY

$DIRECTORY      DK:EXAMPL.*

 21-JUN-77
EXAMPL.BAK    2 14-JUN-77       EXAMPL.BAT      2 21-JUN-77
EXAMPL.CTL    3 21-JUN-77
 3 FILES, 7 BLOCKS
 1903 FREE BLOCKS

$MESSAGE        END OF THE EXAMPLE BATCH STREAM

$EOJ
```

A.5 RT–11 MODE

RT–11 mode lets you enter commands to the RT–11 monitor or to system programs, and lets you create BATCH programs. You can enter RT–11 mode with either the $JOB/RT11 command or the $RT11 command. If you enter RT–11 mode with the $JOB/RT11 command, RT11 mode remains in effect until BATCH encounters the next $JOB command. If you enter RT–11 mode with the $RT11 command, RT–11 mode is in effect until BATCH encounters a $ in the first position of the command line.

When the characters ., $, *, and tab or space appear in the first position of a line (or card column 1), they are control characters and indicate the following:

Command to the RT–11 monitor, for example,

```
.R PIP
```

* Data line; any line not intended to go to the RT–11 monitor or to the BATCH run-time handler, such as a command to the RT–11 PIP program:

```
*FILE1.DAT/D
```

NOTE

BATCH does not pass the * as data to the program. Comment lines (!) cannot appear on data lines, as BATCH would consider them as data.

$ BATCH command. It causes an exit from RT–11 mode if you entered RT–11 mode with the $RT11 command. For example:

```
$RT11                    !ENTER RT-11 MODE
.R PIP
*FILE1.DAT/D
$FORTRAN                 !LEAVE RT-11 MODE
```

space/tab Separator to indicate a line directed to BATCH run-time handler. This separator is indicated by a <TAB> in the following descriptions.

A.5.1 Communicating with RT–11

The most common use of RT–11 mode is to send commands to the RT–11 monitor and to run system programs. For example, you can insert the following commands in the BATCH stream to run PIP and save backup copies of files on DECtape:

```
$RT11
.R PIP
*DT1:*.*=*.FOR
```

You must anticipate and include in the BATCH input stream responses that the called program requires, such as the Y response to DUP's Are you sure? query. Place a line in your BATCH file consisting of Y and RETURN or use the DUP /Y option to suppress the query. For example:

```
$RT11
.INITIALIZE RK1:
*Y
```

You can communicate directly with the RT–11 monitor by using the keyboard monitor commands that are described in Section 4.5 of the *RT–11 System User's Guide*. For example:

```
$RT11
.DELETE/NOQUERY DX1:*.MAC
```

This command deletes all files with a file type of .MAC from device DX1:.

You cannot mix BATCH standard commands with RT–11 mode data lines (lines beginning with an asterisk). For example, the proper way to do a $MOUNT within a sequence of RT–11 mode data commands is:

```
$JOB/RT11
.R MACRO
*A1=A1
*A2=A2
$MOUNT DT0:/PHYSICAL
.R MACRO
*B1=DT:B1
*B2=DT:B2
```

A.5.2 Creating RT–11 Mode BATCH Programs

Advanced system programmers can use RT–11 mode to create BATCH programs. These BATCH programs consist of standard RT–11 mode commands (monitor commands, data lines for input to system programs, etc.) plus special RT–11 mode commands. The BATCH run-time handler interprets these special commands to allow dynamic calculations and conditional execution of the RT–11 mode standard commands. The following can help you create BATCH programs and dynamically control their execution at run-time:

- Labels
- Variable modification:
  1. Equating a variable to a constant or character (LET statement)
  2. Passing the value of a variable to a program
  3. Incrementing the value of a variable by 1
  4. Conditional transfers on comparison of variable values with numeric or character values (IF and GOTO statements)
- Commands to control terminal I/O
- Other control characters
- Comments

A.5.2.1 Labels – You define labels in RT–11 mode to provide a symbolic means of referring to a specific location within a BATCH program. If present, a label must begin in the first character position, must be unique within the first six characters, and must terminate with a colon (:) and a carriage return/line feed combination.

A.5.2.2 Variables – A variable in RT–11 mode is a symbol representing a value that can change during program execution. The 26 variables BATCH permits in a BATCH program have the names A–Z; each variable requires one byte of physical storage. There are four ways to modify variables.

You can assign values to variables in a LET statement.

You can then test these values by an IF statement to control the direction of program execution.

Assign values to variables with a LET statement of the following form:

<TAB>LET x = "c

where:

x represents a variable name in the range A–Z

"c indicates the ASCII value of a character

For example:

```
TABLET A="0
```

This example indicates that the value of variable A is the 7-bit ASCII value of the character 0 (60).

The LET statement can also specify an octal value in the form:

<TAB>LET A = n

where:

n represents an 8-bit signed octal value in the range 0–377. Positive numbers range from 0–177; negative numbers range from 200–377 (–200 to –1).

You can use variables to introduce control characters, such as ESCAPE, into a BATCH stream. For example, wherever 'A' appears in the following BATCH stream, BATCH substitutes the contents of variable A (the code for an ESCAPE):

```
$JOB/RT11
   LET A=33
   !A IS AN ESCAPE
.R EDIT
*EBFILE.MAC'A''A'
*R'A''A'
   !EDIT FILE TO CHANGE THE VERSION NUMBER TO 2
*GVERSION='A'DI2'A''A'
*EX'A''A'
```

Increment the value of a variable by 1 by placing a percentage sign (%) before the variable. For example:

```
TAB%A
```

This command indicates that BATCH must increase the unsigned contents of variable A by 1.

Indicate with an IF statement conditional transfers of control according to the value of a variable. The IF statement has the syntax:

<TAB>IF(x-"c) label1, label2, label3

or

<TAB>IF(x-n) label1, label2, label3

where:

x	represents the variable to be tested
"c	is the ASCII value to be compared with the contents of the variable
n	is an octal integer in the range 0–377
label1 label2 label3	represent the names of labels included in the BATCH stream

When BATCH evaluates the expression (x-"c) or (x-n), the BATCH run-time handler transfers control to:

- label1 if the value of the expression is less than zero
- label2 if the value of the expression is equal to zero
- label3 if the value of the expression is greater than zero

If you omit one of the labels, and the condition is met for the omitted label, control transfers to the line following the IF statement.

NOTE

Since this comparison is a signed byte comparison, 377 is considered to be –1.

The characters + and – allow you to control where BATCH begins searching for label1, label2, and label3. If you precede the label by a minus sign (–), BATCH starts the label search just after the $JOB command. If a plus sign (+) or no sign precedes the label, the label search starts after the IF statement. For example:

```
TAB IF(B-"9) -LOOP, LOOP1,
```

This statement transfers program control to the label LOOP following the $JOB command if the contents of variable B are less than the ASCII value of 9. It transfers control to the label LOOP1 following the IF statement if B is equal to ASCII 9. If the contents of variable B are greater than the ASCII value of 9, program control goes to the next BATCH statement in sequence.

The GOTO statement unconditionally transfers program control to a label you specify as the argument of the statement. You can use one of the following three forms of this statement:

<TAB>GOTO label	Transfers control to the first occurrence of label that appears after this GOTO statement in the BATCH stream
<TAB>GOTO +label	Same as GOTO label

<TAB>GOTO -label	Transfers control to the first occurrence of label that appears after the $JOB command

The following GOTO statement transfers control unconditionally to the next label LOOP if such a label appears in the BATCH stream following the GOTO statement.

```
TABGOTO LOOP
```

NOTE

If BATCH cannot find a label (for example, you unintentionally omit a minus sign), the BATCH handler searches until it reaches the end of the .CTL file and ends the job.

A.5.2.3 Terminal I/O Control — You can issue commands directly to the BATCH run-time handler to control logging console terminal input and output. If you do not enter any of the following commands, BATCH assumes TTYOUT (this includes indirect command files).

<TAB>NOTTY	Does not write terminal input and output to the log file. Comments to the log are still logged.
<TAB>TTYIN	Writes only terminal input to the log file.
<TAB>TTYIO	Writes terminal input and output to the log file. (You should enter this command if using RT–11 mode so that RT–11 mode commands go to the log file.)
<TAB>TTYOUT	Writes only terminal output to the log file (default).

A.5.2.4 Other Control Characters — The system permits other control characters in an RT–11 mode command that begins with a period (.) or an asterisk (*). Following are these control characters and their meanings:

'text'	command to BATCH run-time handler, where text can be one of the following:	
	CTY	accepts input from the console terminal; notifies the operator that action is required by ringing a bell and printing a question mark (?).
	FF	outputs the current log buffer.
	NL	inserts a new line (line feed) in the BATCH stream.
	x	inserts the contents of a variable where x is an alphanumeric variable in the range A through Z. It indicates that BATCH should insert the contents of the variable as an ASCII character at this place in the command string.

"message" directs the message to the console terminal.

The following commands allow the operator to enter the name of a MACRO program to be assembled. The BATCH stream contains:

```
$JOB/RT11
.R MACRO
*'"ENTER MACRO COMMAND STRING"''CTY'
```

The operator receives the following message at the terminal and types a response, followed by a carriage return; BATCH processing continues.

```
ENTER MACRO COMMAND STRING
?FILE,FILE=FILE
```

To run the same BATCH file on several systems with different configurations you need to assign a device dynamically. The following RT–11 mode command lets you request that the listing device name be entered by the operator.

```
 ASSIGN '"PLEASE TYPE LST DEVICE NAME"''CTY'LST
```

The operator receives the message and responds with the device to be used as the listing device (DT2:).

```
PLEASE TYPE LST DEVICE NAME
?DT2:
```

A.5.2.5 Comments – You can include comments in RT–11 mode as separate comment statements. Include comments by typing a separator followed by a ! and the comment. For example:

```
TAB!OPERATOR ACTION IS REQUESTED IN THIS JOB. BE PREPARED.
```

A.5.3 RT–11 Mode Examples

The following are examples of BATCH programs using the RT–11 mode.

This BATCH program assembles, lists, and maps 10 programs with only 12 BATCH commands.

```
$JOB/RT11     !ASSEMBLE, LIST, MAP PROG0 to PROG9
   TTYIO
   !WRITE TERMINAL I/O TO THE LOG FILE
   LET N="0
   !START AT FILE PROG0
LOOP:
.R MACRO
*PROG'N',LOG:/C=PROG'N'/N:TTM
.R LINK
*,LOG:=PROG'N'
   %N
   !INCREMENT VARIABLE N
   IF(N-"9)-LOOP,-LOOP,END
   !TEST FOR END
END:
$EOJ
```

The following program lets you set up a master control stream to run several BATCH jobs with one call to BATCH. First set up a BATCH job (INIT.BAT) that performs a $CHAIN to the master control stream:

```
$JOB/RT11
   LET I="0
   !INITIALIZE INDEX
$CHAIN MASTER        !GO TO MASTER
$EOJ
```

The following is the master control stream (MASTER.BAT) to which INIT chains.

```
$JOB/RT11            !MASTER CONTROL STREAM
   %I
   !BUMP INDEX BY 1
   IF(I-"7),,END
.R BATCH
   !THIS IS A $CHAIN
*JOB'I'
   !RUNS JOB1-JOB7
END:
$MESSAGE END OF BATCH RUN
$EOJ
```

Each job MASTER.BAT will run must contain the following:

```
$JOB
   !BATCH COMMANDS
$CHAIN MASTER
$EOJ
```

Activate the master control stream by calling BATCH as follows:

```
.R BATCH
*INIT
```

A.6 Creating BATCH Programs on Punched Cards

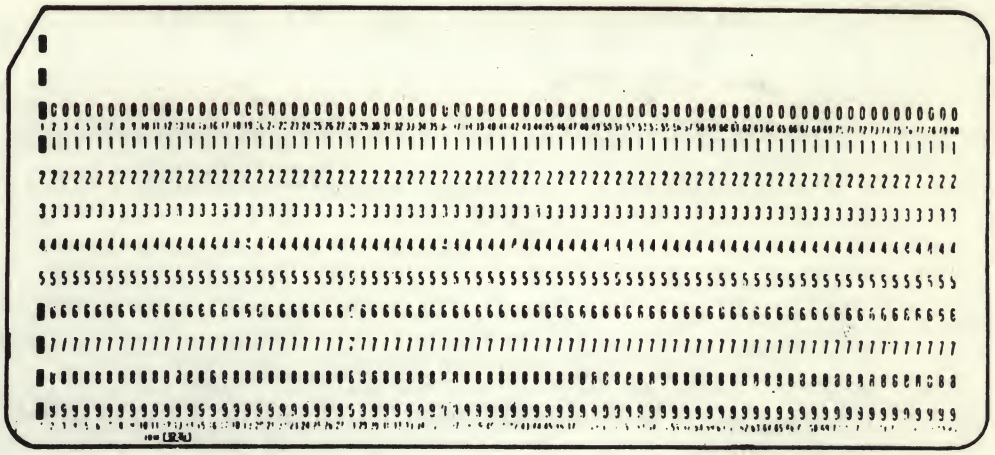
To create a BATCH program on punched cards, punch into the cards the commands described in Section A.4. Each command line occupies a single punched card. Only one card, the EOF card, is different from the standard BATCH commands. The EOF (end-of-file) card terminates the list of jobs from the card reader.

To create the EOF card, hold the MULT PCH key on the keypunch keyboard while typing the following characters:

– & 0 1 6 7 8 9

This procedure produces an EOF card with holes punched in the first column (see Figure A–1).

Figure A–1: EOF Card



To run multiple jobs from the card reader, simply combine the jobs into a single card deck. Make sure that each job has its own $JOB and $EOJ card, and then follow the last $EOJ card with two EOF cards.

Although in general, you terminate BATCH jobs on cards by placing two EOF cards after the last $EOJ card, some card readers may require that you type —F followed by a carriage return. Put two EOF cards and a blank card in the reader and make sure that the card reader is ready. Note that a small card deck (less than 512 characters) may require more than two EOF cards to terminate the deck.

A.7 Operating Procedures

This section describes the operations you must perform to prepare for using BATCH, and for running BATCH.

A.7.1 Loading BATCH

After you bootstrap the RT–11 system and enter the date and time, you must make the BATCH run-time handler resident by typing the RT–11 LOAD command as follows:

```
LOAD BA:
```

You detach and unload the BATCH run-time handler with the /U option in the BATCH compiler command line (see Section A.7.2).

NOTE

If BATCH crashes, you must unload BATCH with the UNLOAD command and then reload BATCH with the LOAD command. This ensures that the BATCH handler is properly initialized when you rerun BATCH.

You must make the BATCH log device and list device resident unless the log or list device is SY:, or unless it is a device for which the handler is already resident. Load the log device, using the following syntax:

.LOAD log-device

where:

log-device — represents the device to which BATCH must write the log file

For example:

```
.LOAD LP:
```

You can, of course, load device handlers with a single LOAD command. For example:

```
.LOAD BA:,LP:
```

You must then assign the logical device name LOG to the log device. Use the RT–11 monitor ASSIGN command in the form:

.ASSIGN log-device LOG

For example, if LP: is the log device, type:

```
. ASSIGN LP LOG
```

Then assign the logical device name LST: using the RT–11 ASSIGN command in the form:

.ASSIGN list-device LST

where:

list-device — represents the physical device BATCH must use for listings

If, for example, you want to produce listings on the line printer, type:

```
ASSIGN LP LST
```

NOTE

Do not use the DEASSIGN command with no arguments in a BATCH program since it deassigns the log and list devices, possibly causing the BATCH job to terminate.

You must also make resident the BATCH run-time handler input device (compiler output device). If this device is already resident or is SY:, you do not need to load it. For example, to load the DECtape handler as the input device, type:

```
LOAD DT
```

If the input file to the BATCH compiler is on cards, load the card reader handler by typing:

```
LOAD CR
```

NOTE

If input is on cards, you must use the RT–11 monitor SET command (before loading the handler) to specify CRLF and NOIMAGE modes. That is, the following command appends a carriage return/line feed combination to each card image.

```
' SET CR CRLF
```

The following command translates the card by packing card code into ASCII data, one column per byte.

```
' SET CR NOIMAGE
```

If card images do not properly translate to ASCII, you may have to change the card translation codes by using one of the following commands:

```
' SET CR CODE=29
```

or

```
' SET CR CODE=26
```

See Section 4.4.

A.7.2 Running BATCH

When you have loaded all necessary handlers, run the BATCH compiler as follows:

```
' R BATCH
```

BATCH responds by printing an asterisk (*) to indicate its readiness to accept commands. In response to the *, type the output file specifications for the control file followed by an equals sign. Then type the input file specifications for the BATCH file as follows:

[[output-filespec][,log-filespec][/option...] =]input-filespec[...,
input-filespec][/option...]

where:

output-filespec is the BATCH compiler output device and file the BATCH run-time handler must use. The device you specify must be random-access. Your BATCH job should not delete or move this file. Your BATCH job

should avoid compressing the system volume with the SQUEEZE command or the DUP /S option. If you omit output-filespec, BATCH generates a file on the default device DK: with the same name as the first input file but with a .CTL file type. If you do not specify a file type in output-filespec, BATCH assumes .CTL.

log-filespec

is the log file created by the BATCH run-time handler. If you do not specify a log device, BATCH assumes LOG:. The device name you specify for log-filespec must be the same as you assign to LOG:.

You can change the size of a log file on a file-structured device from the default size of 64(decimal) blocks. To make this change, enclose the required size in square brackets. For example:

```
*, FILE,LOG[10]=FILE
```

The default file type for the log-filespec is .LOG.

input-filespec

represents an input file. If you do not specify a file type, BATCH assumes .BAT. If you specify a .CTL file, BATCH assumes a precompiled file that must be the only file in the input list.

/option

is an option from the following list:

/N

compiles but does not execute. This option creates a BATCH control file (.CTL), generates an ABORT JOB message at the beginning of the log file, and returns to the RT–11 monitor.

/T:n

if n = 0, sets the /NOTIME option as the default on the $JOB command. If n = 1, the default option on the $JOB command is /TIME.

/U

indicates that the BATCH compiler must detach the BATCH run-time handler from the RT–11 monitor and unload the handler.

NOTE

You need not specify the RT–11 monitor UNLOAD BA command to remove the handler. Specifying /U to BATCH causes the handler to detach and unload.

/X indicates that the input is a precompiled BATCH program. Use this option when you do not specify the .CTL file type.

<RET> prints the version number of the BATCH compiler.

The following example calls BATCH to compile and execute three input files (PROG1.BAT, PROG2.BAT, PROG3.BAT) to generate on DK: the compiler output files, and to generate on LOG: a log file.

```
.R BATCH
*PROG1.BAT,PROG2.BAT,PROG3.BAT
```

The following commands print the version number of BATCH, then compile and run SYBILD.BAT.

```
.R BATCH
*<RET>
BATCH V04.00A
*SYBILD
```

The following commands compile PROTO.BAT to create PROTO.CTL but do not run the compiled BATCH stream.

```
.R BATCH
*PROTO/N
```

Type the following commands to unlink BA.SYS from the monitor and to unload it.

```
.R BATCH
*/U
```

The following commands compile FILE.BAT from magtape to create FILE.CTL on RK1:. They execute the compiled file and create a log file named FILE.LOG (of size 20) on LOG:.

```
.R BATCH
*RK1:FILE,FILE[20]=MT:FILE
```

The following commands execute a precompiled job called FILE.TST.

```
.R BATCH
*FILE.TST/X
```

The following commands execute a precompiled job called FILE.CTL.

```
.R BATCH
*FILE/X
```

The following commands accept input from the card reader to create a file called TEMP.CTL. BATCH stores this file on DK: and executes it.

```
. R BATCH
* CR:
```

The following commands accept input from the card reader to create a file called JOB.CTL. BATCH stores the file on DK: and executes it.

```
. R BATCH
* JOB=CR:
```

A.7.3 Communicating with BATCH Jobs

During the execution of a BATCH stream, BATCH can request the operator to service a peripheral device, to provide information, or to insert a command line into the BATCH stream. The operator does this by typing directives to the BATCH handler on the console terminal.

NOTE

> These directives are equivalent to the compiler output that BATCH generates in the .CTL file. The .CTL file is an ASCII file that you can list by using the PRINT or TYPE commands or by running PIP.

These directives have the form:

\dir

where:

dir represents one of the directives listed in Table A–6

To use these directives, the operator must get control of the BATCH runtime handler. This can be achieved through a /WAIT or a CTY in the BATCH stream, or by typing a carriage return on the console terminal. If a carriage return is typed, the operator does not know exactly where the BATCH stream has been interrupted. When BATCH executes a command, it acknowledges the carriage return and prints a carriage return/line feed combination at the terminal. The operator can then enter a directive from Table A–6. The most useful directives are marked with an asterisk (*). Some directives are not particularly useful in this mode, but are listed to explain completely the BATCH compiler output.

In the following example, the operator must interrupt the BATCH handler to enter information from the console. As a result of a /WAIT or 'CTY' in the BATCH stream, the following message appears at the terminal:

```
$MESSAGE/WAIT WRITE NECESSARY FILES TO DISK
```

Table A–6: Operator Directives to BATCH Run-Time Handler

Directive	Function
\@	Sends the characters that follow to the console terminal.
*\A	Changes the input source to be the console terminal.
*\B	Changes the input source to be the BATCH stream.
*\C	Sends the following characters to the log device.
*\D	Considers the following characters as user data.
*\E	Sends the following characters to the RT–11 monitor.
*\F	Forces the output of the current log block. If this directive is followed by any characters other than another BATCH backslash (\) directive, the BATCH job prints an error message and terminates. BATCH then returns control to the RT–11 monitor.
\G	Gets characters from the console terminal until a carriage return is encountered.
\Hn	Help function that changes the logging mode. n specifies the following: 0 Log only .TTYOUT and .PRINT 1 Log .TTYOUT, .PRINT, and .TTYIN 2 Do not log .TTYOUT, .PRINT, and .TTYIN 3 Log only .TTYIN
\Ivxlabel1? label2? label3?	IF statement that causes conditional transfer, where v is a variable name in the range A-Z; x is a value for the signed 8-bit comparison (v-x); and label1, label2, label3 are 6-character labels to which control is transferred under certain conditions. (All labels must be six characters in length; if too short, pad with spaces.) If v-x is less than 0, control transfers to label1; if v-x is equal to 0, control goes to label2; if v-x is greater than 0, control goes to label3. The direction for the label search is indicated by ?; if ? is 0, the search begins at the beginning of this job; if ? is 1, the label search begins after the IF statement.
\Jlabel?	Jump, unconditional transfer; where label is a 6-character label and ? is 0 or 1. (All labels must be six characters in length; if too short, pad with spaces.) If ? = 0, label is a backward reference; if ? = 1, label is a forward reference.
\Kv0	Increment variable v, where v is a variable name in the range A-Z.
\Kvln	Stores the 8-bit number n in variable v.
\Kv2	Takes the value in variable v and returns it to the program (via .TTYIN).
\Llabel	Inserts label as a 6-character alphanumeric string in the BATCH stream. (All labels must be six characters in length; if too short, pad with spaces.) Labels must not include backslash characters. Characters beyond six are ignored.

To divert BATCH stream input from the current file to the console terminal, the operator types \E, enters commands to the RT-11 monitor, then types \B. Control then returns to the BATCH stream. The following example illustrates this procedure.

```
. R BATCH
* NEXT
 WRITE NECESSARY FILES TO DISK
? \A\E

\   COPY DT1:FILE.MAC RK:

 FILES COPIED:
DT1:FILE.MAC    TO RK:FILE.MAC

 \E\F\B

2
END BATCH
```

The following BATCH program lets you make frequent edits to a file and list only the edits. First, create a BATCH program that assembles with a listing and link the file. This BATCH program, called COMPIL.BAT, contains:

```
$JOB/RT11
   TTYIO
   !WRITE TERMINAL I/O TO LOG FILE
.R MACRO
   !CALL THE MACRO ASSEMBLER
*FILE,FILE/C=FILE
$MESSAGE/WAIT OK TO TYPE EDIT COMMANDS
.R LINK
   !CALL THE RT-11 LINKER
*FILE,LOG:=FILE
$EOJ
```

At run time, you can insert commands into the BATCH stream from the console terminal. These commands search for the section of the listing file that has been edited, then list this section to the log. You must insert the command after the R MACRO command but before the R LINK command. The following example illustrates this procedure.

```
. R BATCH
* COMPIL
 OK TO TYPE EDIT COMMANDS
? \A\E

\ ER EDIT

* ERFILE.LST$$
* EWFILE.SEC$$
* PRETRY:$=J$$
* \L$$
RETRY:  0                          ;HIGH ORDER BIT USED FOR "RESET IN PROGRESS FLAG
     49 000020  016705  177764                MOV     RKCQE,R5          ;GET Q P
     50 000024  011502                        MOV     @R5,R2            ;R2 = BL
     51 000026  016504  000002                MOV     2(R5),R4          ;R4 = UN
     52 000032  006204                        ASR     R4                ;ISOLATE
```

```
     53 000034  006204                          ASR     R4
     54 000036  006204                          ASR     R4
     55 000040  000304                          SWAB    R4
     56 000042  042704  017777                  BIC     #^C<160000>,R4
     57 000046  000404                          BR      2$                  ;ENTER C
*EX$$

E\C\B

END BATCH
```

A.7.4 Terminating BATCH

When BATCH terminates normally, it prints the following message and returns control to the RT–11 monitor:

```
END BATCH
```

To abort BATCH while it is executing a BATCH stream, interrupt the BATCH handler by typing a carriage return. When BATCH executes the next command after the carriage return, it prints a carriage return/line feed combination at the console terminal. You then gain control of the system. Type \F followed by a carriage return. The BATCH handler responds with the FE (forced exit) error message and writes the remainder of the log buffer. Control returns to the RT-11 monitor.

Typing two CTRL/Cs terminates BATCH immediately. Use two CTRL/Cs when BATCH is in a loop or when a long assembly is running. In these cases, BATCH responds slowly to your carriage return interrupt.

A.8 Differences Between RT–11 BATCH and RSX–11D BATCH

Some programmers run their RT–11 BATCH programs under RSX–11D. Note the differences between the two BATCH implementations listed in Table A–7. BATCH programs that run under both systems must be compatible with both RT–11 and RSX–11D BATCH.

Table A–7: Differences Between RT–11 and RSX–11D BATCH

Characteristic	RT–11	RSX–11D
File descriptors	filespec/option	SY:filnam.typ/option
Default listing file type	.LST(or .LIS)	.LIS
Executable file type	.SAV	.EXE
Incompatible commands	$BASIC $CALL $CHAIN $LIBRARY $RT11 $SEQUENCE	$MCR

(Continued on next page)

Table A–7: Differences Between RT–11 and RSX–11D BATCH (Cont.)

Characteristic	RT–11	RSX–11D
Incompatible options	$COPY/DELETE $CREATE/DOLLARS $CREATE/LIST $DATA/DOLLARS $DATA/LIST $DIR file/LIST $DISMOUNT/WAIT $DISMOUNT lun:/LOGICAL $FORTRAN/DOLLARS $FORTRAN/MAP $JOB/BANNER $JOB/LIST $JOB/RT11 $JOB/TIME $JOB/UNIQUE $LINK/LIBRARY $LINK/OBJECT $MACRO/CREF $MACRO/DOLLARS $MACRO/LIBRARY $MACRO/MAP $MESSAGE/WAIT $MESSAGE/WRITE $PRINT/DELETE	 $DIR file/DIRECTORY $JOB/NAME $JOB/LIMIT $JOB/MCR $LINK/MCR
$DATA input	Appears as if from input	Appears as if from a file named FOR001.DAT
Logical device names	In $MOUNT and $DISMOUNT	Logical unit numbers only
$RUN	You must specify file name	RSX11DBAT.EXE is default

Appendix B
System Utility Program Options and Monitor Command Equivalents

Table B–1 lists all the system programs and options with their keyboard monitor equivalents. Note that some system program options are inaccessible through any keyboard monitor command; each is marked by an asterisk.

In this table:

- The first column lists the system program.
- The second column lists the system program's option.
- The third column lists the keyboard monitor command by which the system program is invoked.
- The fourth column lists the monitor equivalent of the system program option.

Note also that some system programs are accessible through more than one keyboard monitor command (for example, DUP is accessed through BOOT, COPY, CREATE, DIRECTORY, INITIALIZE, and SQUEEZE).

Table B–1: System Program/Monitor Command Equivalents

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
BINCOM			
	/B	DIFFERENCES/BINARY	/BYTES
	/D	DIFFERENCES/BINARY	/DEVICE
	/E:n	DIFFERENCES/BINARY	/END[:n]
	/H	*	
	/O	DIFFERENCES/BINARY	/ALWAYS
	/Q	DIFFERENCES/BINARY	/QUIET
	/S:n	DIFFERENCES/BINARY	/START[:n]
BUP		BACKUP	–
	/I	BACKUP	/DEVICE
	/L	DIRECTORY	/BACKUP
	/X	BACKUP	/RESTORE
	/Y	BACKUP	/NOQUERY
	/Z	INITIALIZE	/BACKUP

(Continued on next page)

Table B–1: System Program/Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
DIR			
	/A	DIRECTORY	/ALPHABETIZE
	/B	DIRECTORY	/BLOCKS (disks)
			/POSITION (magtapes)
	/C:n	DIRECTORY	/COLUMNS:n
	/D[:date]	DIRECTORY	/DATE[:date]
	/E	DIRECTORY	/FULL
	/F	DIRECTORY	/FAST
	/G	DIRECTORY	/BEGIN
	/J[:date]	DIRECTORY	/SINCE[:date]
	/K[:date]	DIRECTORY	/BEFORE[:date]
	/L	*	
	/M	DIRECTORY	/FREE
	/N	DIRECTORY	/SUMMARY
	/O	DIRECTORY	/OCTAL
	/P	DIRECTORY	/EXCLUDE
	/Q	DIRECTORY	/DELETED
	/R	DIRECTORY	/REVERSE
	/S[:xxx]	DIRECTORY	/SORT[:category]
	/T	DIRECTORY	/PROTECTION
	/U	DIRECTORY	/NOPROTECTION
	/V[:ONL]	DIRECTORY	/VOLUMEID[:ONLY]
DUMP			
	/B	DUMP	/BYTES
	/E:n	DUMP	/END:n
	/G	DUMP	/IGNORE
	/N	DUMP	/NOASCII
	/O:n	DUMP	/ONLY:n
	/S:n	DUMP	/START:n
	/T	DUMP	/FOREIGN
	/W	DUMP	/WORDS
	/X	DUMP	/RAD50
DUP			
	/B[:RET]	INITIALIZE	/BADBLOCKS[:RET]
	/C	CREATE	
	/D	INITIALIZE	/RESTORE
	/E:n	COPY	/END:n
	/F	COPY, DIRECTORY	/FILES
	/G:n	COPY, CREATE	/START:n
	/H		*
	/I	COPY	/DEVICE
	/L	ASSIGN, DEASSIGN	
	/K	DIRECTORY	/BADBLOCKS
	/N:n	INITIALIZE	/SEGMENTS:n
	/O	BOOT	
	/Q	BOOT	/FOREIGN
	/R[:RET]	COPY	/RETAIN
		INITIALIZE	/REPLACE[:RETAIN]
	/S	SQUEEZE	
	/T:n	CREATE	/EXTENSION:n

(Continued on next page)

Table B–1: System Program/Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
	/U[:xx]	COPY	/BOOT[:val]
	/V[:ONL]	INITIALIZE	/VOLUMEID[:ONLY]
	/W	DIRECTORY, COPY, INITIALIZE, SQUEEZE, BOOT	/WAIT
	/X	*	
	/Y	COPY, INITIALIZE, SQUEEZE	/NOQUERY
	/Z[:n]	INITIALIZE	
ERROUT			
	/A	SHOW ERRORS	/ALL
	/F:date	SHOW ERRORS	/FROM:date
	/S	SHOW ERRORS	/SUMMARY
	/T:date	SHOW ERRORS	/TO:date
FILEX			
	/A	COPY	/ASCII
	/D	DELETE	
	/F	DIRECTORY	/FAST
	/I	COPY	/IMAGE
	/L	DIRECTORY	
	/P	COPY	/PACKED
	/S	COPY	/DOS
	/T	COPY	/TOPS
	/U[:n.]	COPY	/INTERCHANGE[:size]
	/V[:ONL]	DIRECTORY, INITIALIZE	/VOLUMEID[:ONLY]
	/W	COPY, DELETE, DIRECTORY, INITIALIZE	/WAIT
	/Y	INITIALIZE	/NOQUERY
	/Z	INITIALIZE	
FORMAT			
	/P:n	FORMAT	/PATTERN:value
	/S	FORMAT	/SINGLEDENSITY
	/V[:ONL]	FORMAT	/VERIFY[:ONLY]
	/W	FORMAT	/WAIT
	/Y	FORMAT	/NOQUERY
IND	Not accessible through keyboard monitor commands		
LD			
	/A:ddd	ASSIGN	–
	/C	SET LDn	CLEAN
	/L:n	MOUNT, DISMOUNT	
	/R:n	MOUNT, DISMOUNT	NOWRITE
	/W:n	MOUNT, DISMOUNT	WRITE
LIBR			
	/A	*	
	/C	LIBRARY	/PROMPT

(Continued on next page)

Table B–1: System Program/Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
	/D	LIBRARY	/DELETE
	/E	LIBRARY	/EXTRACT
	/G	LIBRARY	/REMOVE
	/M[:n]	LIBRARY	/MACRO[:n]
	/N	*	
	/P	*	
	/R	LIBRARY	/REPLACE
	/U	LIBRARY	/UPDATE
	/W	*	
	/X	*	
	//	*	
LINK			
	/A	LINK	/ALPHABETIZE
	/B:n	LINK	/BOTTOM:value
	/C	LINK	/PROMPT
	/D	LINK, EXECUTE	/DUPLICATE
	/E:n	LINK	/EXTEND:n
	/F	*	
	/G	*	
	/H:n	LINK	/TOP[:value]
	/I	LINK	/INCLUDE
	/K:n	LINK	/LIMIT:n
	/L	LINK	/LDA
	/M[:n]	LINK	/STACK[:value]
	/N	LINK, EXECUTE	/GLOBAL
	/O:n	*	
	/P:n	*	
	/Q	*	
	/R[:n]	LINK	/FOREGROUND[:STACKSIZE]
	/S	LINK	/SLOWLY
	/T[:n]	LINK	/TRANSFER[:value]
	/U:n	LINK	/ROUND:n
	/V:n[:m]	LINK	/XM
	/W	LINK	/WIDE
	/X	LINK	/NOBITMAP
	/Y:n	LINK	/BOUNDARY:value
	/Z:n	LINK	/FILL:n
	//	*	
MACRO			
	/C:arg	MACRO	/CROSSREFERENCE[:type[...:type]]
	/D:arg	MACRO	/DISABLE[:type[...:type]]
	/E:arg	MACRO	/ENABLE[:type[...:type]]
	/L:arg	MACRO	/SHOW:type
	/M	MACRO	/LIBRARY
	/N:arg	MACRO	/NOSHOW:type
ODT	Not accessible through keyboard monitor commands		
PAT	Not accessible through keyboard monitor commands		

(Continued on next page)

Table B–1: System Program/Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
PIP			
	/A	COPY	/ASCII
	/B	COPY	/BINARY
	/C[:date]	COPY, DELETE, PRINT PROTECT, RENAME TYPE, UNPROTECT	/DATE[:date], /NEWFILES
	/D	DELETE	–
		PRINT, TYPE	/DELETE
	/E	COPY, DELETE, PRINT, PROTECT, RENAME TYPE, UNPROTECT	/WAIT
	/F	COPY, RENAME	/PROTECTION
		PROTECT	–
	/G	COPY	/IGNORE
	/H	COPY	/VERIFY
	/I[:date]	COPY, DELETE, PRINT, PROTECT, RENAME TYPE, UNPROTECT	/SINCE[:date]
	/J[:date]	COPY, DELETE, PRINT, PROTECT, RENAME TYPE, UNPROTECT	/BEFORE[:date]
	/K:n	PRINT, TYPE	/COPIES:n
	/M:n	COPY, DELETE	/POSITION:n
	/N	COPY, RENAME	/NOREPLACE
	/O	COPY	/PREDELETE
	/P	COPY, DELETE PROTECT, UNPROTECT	/EXCLUDE
	/Q	COPY, DELETE, PRINT PROTECT, RENAME TYPE, UNPROTECT	/QUERY
	/R	RENAME	–
	/S	COPY	/SLOWLY
	/T[:date]	COPY, PROTECT RENAME, UNPROTECT	/SETDATE[:date]
	/U	COPY	/CONCATENATE
	/V	COPY	/MULTIVOLUME
	/W	COPY, DELETE, PRINT PROTECT, RENAME TYPE, UNPROTECT	/LOG
	/X	COPY, DELETE, PRINT PROTECT, RENAME TYPE, UNPROTECT	/INFORMATION
	/Y	COPY, DELETE, PRINT PROTECT, RENAME, TYPE, UNPROTECT	/SYSTEM
	/Z	COPY, RENAME	/NOPROTECTION
		UNPROTECT	–
QUEMAN			
	/A	*	
	/C[:date]	PRINT	/DATE[:date], /NEWFILES
	/D	PRINT	/DELETE

(Continued on next page)

Table B–1: System Program/Monitor Command Equivalents (Cont.)

System Program	Option	Keyboard Monitor Command	Keyboard Monitor Option
	/H:n	PRINT	/FLAGPAGE:n
	/I[:date]	PRINT	/SINCE[:date]
	/J[:date]	PRINT	/BEFORE[:date]
	/K:n	PRINT	/COPIES:n
	/L	SHOW	QUEUE
	/M	DELETE	/ENTRY
	/N	PRINT	/NOFLAGPAGE
	/P	*	
	/Q	PRINT	/QUERY
	/R	*	
	/S	*	
	/W	PRINT	/LOG
	/X	PRINT	/INFORMATION
	//	PRINT	/PROMPT
RESORC			
	/A	SHOW	ALL
	/C	*	
	[dev:]/D	SHOW	DEVICES[dd:]
	/H	*	
	/J	SHOW	JOBS
	/L	SHOW	
	/M	*	
	/O	*	
	/S	SHOW	SUBSET
	/T	SHOW	TERMINALS
	/X	SHOW	MEMORY
	/Z	SHOW	CONFIGURATION
SIPP	Not accessible through keyboard monitor commands		
SLP	Not accessible through keyboard monitor commands		
SRCCOM			
	/A	DIFFERENCES	/AUDITTRAIL
	/B	DIFFERENCES	/BLANKLINES
	/C	DIFFERENCES	/NOCOMMENTS
	/D	DIFFERENCES	/CHANGEBAR
	/F	DIFFERENCES	/FORMFEED
	/L:[n]	DIFFERENCES	/MATCH:[n]
	/S	DIFFERENCES	/NOSPACES
	/T	DIFFERENCES	/NOTRIM
	/V:i:d	*	

INDEX

/A
  DIR option, 4–2
  Error Logger option, 16–8
  FILEX option, 7–3
  LD option, 9–2
  LIBR option, 10–5
  LINK option, 11–43
  PIP option, 13–8
  QUEMAN option, 17–4
  RESORC option, 14–2
  SIPP option, 20–3
  SLP option, 21–3
ABS
  p-sect attribute, 11–5
Absolute block parameters (table), 11–17
Absolute load module
  creating, 11–16
Absolute program section
  attributes (table), 11–6
  contents of, 11–3
Access code
  p-sect attributes, 11–5
Active page register
  definition of, 11–30
Alloc-code
  function of, 11–4
  p-sect attributes, 11–5
APR
  See Active page register
ASCII files
  copying, 13–8
ASECT
  See Absolute program section
Audit trail
  SLP, 21–5
    disabling, 21–5
    enabling, 21–5
    specifying, 15–8
  specifying size of, 21–3

/B
  BINCOM option, 2–3
  DIR option, 4–4
  DUP option, 6–19
  LINK option, 11–43
  PIP option, 13–9
  SLP option, 21–3
Background jobs
  debugging with ODT, 18–21
Backup utility program
  See BUP
Backup volumes (BUP)
  initializing, 3–8
Bad block replacement, 6–17
Bad blocks
  covering, 6–19
  replacing, 6–17
Bad block scans
  including name of files with bad blocks, 6–9
  performing, 6–7
.BAD files
  copying, 13–3
  deleting, 13–3
  PIP treatment of, 13–3

Banner pages
printing specified number of, 17–5
setting default number of, 17–7
suppressing printing of, 17–7
$BASIC
BATCH command, A–13
BATCH, A–1 to A–52
assembling MACRO source files using, A–26
calling another BATCH control file, A–14
calling BASIC with, A–13
calling FORTRAN compiler with, A–21
character set, A–8
character set (table), A–9
command field options, A–3
command field options (table), A–4
command fields in control statement, A–2
command line syntax, A–2
command names, A–3
commands, A–11 to A–33
commands (table), A–12
comments, A–7
communication with RT–11, A–32, A–36
compiler, A–1
creating files with, A–16
creating programs for on punched cards, A–42
data transfers, A–17
with FORTRAN programs, A–18
deassigning logical device names with, A–19
directory operation, A–19
executing a program with, A–32
file deletion, A–18
file specifications in control statements to, A–6
file types, A–7
function of, A–1
general rules, A–11
hardware requirements, A–1
image mode copy, A–15
indicating beginning of job for, A–23
indicating end of a job for, A–20
indicating end-of-data for, A–20
jobs
assigning identification number to, A–33
communicating with, A–48
link operations, A–25
loading, A–43
nesting control files with BATCH, A–14
preparing to use, A–43
printing files with, A–31
RT–11
differences from RSX–11D, A–51
RT–11 mode, A–32, A–35
RT–11 mode (examples), A–41
RT–11 mode programs
comments in, A–41
creating, A–37
running, A–45
run-time handler, A–1
software requirements, A–2
specification field options, A–7
specification field options (table), A–8
specification fields, A–3
specifying devices in control statement for, A–6
stream
running SIPP from, 20–17
stream (example), A–33
temporary files, A–10
terminal I/O control with, A–40
terminating, A–51
wildcards
using with, A–6
Binary comparison program
See BINCOM
Binary files
comparing, 2–1
copying, 13–9
BINCOM, 2–1 to 2–6
byte-by-byte comparison, 2–3
calling, 2–1
command syntax, 2–2
device comparison, 2–3
differences file
forcing creation of, 2–3
format, 2–4
suppressing, 2–4
end block for comparison, 2–3
examples, 2–5
function of, 2–1
halting, 2–1
help option, 2–3
options, 2–3, 2–4
and keyboard command equivalents (table), B–1
options (table), 2–3
output, 2–1, 2–4
processing of files, 2–1
SIPP command file as output from, 2–5, 2–6
forcing creation of, 2–3
starting block for comparison, 2–4
wildcards with, 2–2

Bitmap, 11–16
  suppressing creation of, 11–56
Bootstrap
  copying to a volume, 6–13
Bootstrapping
  foreign volumes, 6–10
Breakpoints
  using with ODT, 18–12, 18–22
Bullets
  in SRCCOM differences listing, 15–7
    changing character used for, 15–4
BUP, 3–1 to 3–9
  calling, 3–1
  command string syntax, 3–1
  default operation, 3–2
  directory listing
    sample for magtape, 3–7
    sample for random-access volume, 3–6
  directory operation, 3–2, 3–6
  function of, 3–1
  image mode backup, 3–2
    command syntax, 3–3, 3–4
    example, 3–4, 3–6
    for files, 3–2
    for volumes, 3–4
  initializing volumes for use with, 3–2, 3–3, 3–8
    magtape, 3–2, 3–3
  magtapes with, 3–2
  options, 3–4 to 3–8
    and keyboard command equivalents (table), B–1
  options (table), 3–2
  restoring files and volumes with, 3–2, 3–7
  suppressing initialization confirmation, 3–2
  terminating, 3–1
  wildcards with, 3–1

/C
  DIR option, 4–4
  DUP option, 6–2
  LD option, 9–3
  LIBR option, 10–6
  LINK option, 11–44
  MACRO option, 12–8
  PAT option, 19–2, 19–8
  PIP option, 13–9
  QUEMAN option, 17–4
  RESORC option, 14–3
  SIPP option, 20–3
  SLP option, 21–3, 21–11
  SRCCOM option, 15–4
$CALL
  BATCH command, A–14
$CHAIN
  BATCH command, A–14
Changebars
  in SRCCOM differences listing, 15–7
    changing character used for, 15–4
Changes
  marking in SRCCOM differences listing, 15–4, 15–7
Checksum
  PAT, 19–2, 19–8
  SIPP, 20–3, 20–15
  SLP, 21–3, 21–11
Command String Interpreter
  See CSI
Comments
  ignoring during SRCCOM comparison, 15–4
  in BATCH files, A–7
COMMON attributes (table), 11–6
COMMON statement (FORTRAN)
  creating p-sects with, 11–4
Comparison
  of binary files
    See BINCOM
  of source files
    See SRCCOM
CON
  p-sect attribute, 11–5
Concatenating files, 13–16
Configuration
  hardware
    displaying, 14–5, 14–10
  software
    displaying, 14–3
Confirmation prompts
  FORMAT
    suppressing, 8–7
  PIP
    requesting, 13–15
  QUEMAN
    requesting when printing files, 17–8
$COPY
  BATCH command, A–15
Copy operations
  reducing errors during, 13–16
  verifying, 13–13
$CREATE
  BATCH command, A–16
Creation date of files, 13–2

CREF table
  contents of, 12–9
  example, 12–11
  generating, 12–8, 12–10
Cross-reference (CREF) listing
  generating, 11–50
  in load map, 11–40
  in load map (illustration), 11–39
Cross-reference (CREF) table
  See CREF table
CSECT
  named, 11–6
  unnamed, 11–6
CSECT attributes (table), 11–6
.CSECT directive, 11–4
CSI
  command string syntax, 1–1, 1–3
    default devices in, 1–2
    radix of numeric arguments, 1–3
  function of, 1–1
CTRL/Y
  with SIPP, 20–12
CTRL/Z
  with SIPP, 20–11

/D
  BINCOM option, 2–3
  DIR option, 4–5
  DUP option, 6–19
  FILEX options, 7–10
  LIBR option, 10–6
  LINK option, 11–45
  MACRO option, 12–6
  PIP option, 13–9
  QUEMAN option, 17–5
  RESORC option, 14–3
  SIPP option, 20–3
  SLP option, 21–3
  SRCCOM option, 15–7
D
  p-sect attribute, 11–5
$DATA
  BATCH command, A–17
DECsystem–10 volumes
  copying to RT–11 volumes, 7–8
$DELETE
  BATCH command, A–18
Deleting files, 13–9
  .BAD, 13–9
  .SYS, 13–9
  with BATCH, A–18
  with FILEX, 7–10
Device assignments
  displaying, 14–3, 14–6
Device comparison
  binary
    See BINCOM
Device failure
  using the Error Logger to predict, 16–1
Device handlers
  status of
    displaying, 14–3
Device utility program
  See DUP
Differences listing
  SRCCOM
    See SRCCOM
DIR, 4–1 to 4–12
  See also Directory listings
  calling, 4–1
  command string syntax, 4–1
  halting, 4–1
  options, 4–2 to 4–11
    and keyboard command equivalents (table), B–2
  options (table), 4–3
  reading listings produced by, 4–2
$DIRECTORY
  BATCH command, A–19
Directory listings
  default format, 4–7
  excluding certain files from, 4–8
  FILEX, 7–9
  including deleted files in, 4–9
  including files created before certain date in, 4–6
  including files created since certain date in, 4–6
  including file starting block numbers, 4–4
  including files with certain date in, 4–5
  including only file names and types in, 4–5
  including protected files in, 4–11
  including unprotected files in, 4–11
  including unused areas in, 4–5, 4–7
  including volume ID and owner name in, 4–11
  reading, 4–2
  sorting, 4–9
    by creation date, 4–10
    by file name, 4–10
    by file type, 4–10
    by position on volume, 4–10
    by size, 4–10
    in alphabetical order, 4–2
    in reverse order, 4–9

specifying number of columns for, 4–4
starting with file you specify, 4–6
summary format, 4–7
with octal sizes and block numbers, 4–8
Directory segments
changing default number of, 6–16
default number of (table), 6–17
determining number of entries in, 6–16
DIRECTORY utility program
See DIR
$DISMOUNT
BATCH command, A–19
DOS/BATCH volumes
copying files to and from, 7–4
Double-density diskettes
formatting in single-density mode, 8–6
DUMP, 5–1 to 5–6
calling, 5–1
command syntax, 5–1
examples, 5–3 to 5–5
halting, 5–1
operations with magtape, 5–2
options
and keyboard command equivalents (table), B–2
options (table), 5–1
DUP, 6–1 to 6–19
bad block scans with, 6–7
bootstrapping volumes, 6–9
foreign, 6–10
calling, 6–1
changing volumes during operations, 6–15
command string syntax, 6–1
copying bootstrap, 6–13
copying to or from magtapes with, 6–6, 6–9
creating files with, 6–2
deleted files
recovering with, 6–4
extending files, 6–12
function of, 6–1
halting, 6–1
image copying volumes with, 6–5
command syntax, 6–6
options, 6–1, 6–2 to 6–19
and keyboard command equivalents (table), B–2
valid combinations of (table), 6–2
options (table), 6–3, 6–4
performing bad block scans with, 6–9
preserving output volume bad block replacement table, 6–6
printing or changing volume ID with, 6–14
query messages
suppressing, 6–15

/E
BINCOM option, 2–3
DIR option, 4–5
LIBR option, 10–7
LINK option, 11–47
MACRO option, 12–6
PIP option, 13–10
EL.SYS
changing size of internal buffer, 16–2
function of, 16–2
ELINIT
function of, 16–3
Entry point
definition of, 11–7
$EOD
BATCH command, A–20
$EOJ
BATCH command, A–20
ERRLOG.DAT
function of, 16–3
initializing, 16–6
specifying device for, 16–6
specifying size for, 16–7
ERRLOG.REL
function of, 16–3
Error codes
MACRO, 12–12
Error Logger, 16–1 to 16–14
analyzing reports generated by, 16–8
cache memory error report, 16–11
calling
with FB and XM monitors, 16–6
with SJ monitor, 16–5
devices that support, 16–1
disabling under SJ monitor, 16–5
file environment and error count report, 16–13
function of, 16–1
generating reports, 16–3, 16–7
halting under FB or XM monitor, 16–6
internal buffer under SJ
changing size of, 16–2
clearing, 16–5
memory parity error report, 16–10
options for generating reports, 16–8
processing under the SJ monitor, 16–2
running under the FB or XM monitor, 16–3
statistics files, 16–3

statistics-gathering under SJ monitor, 16–5
storage device error report, 16–9
summary error report for memory statistics, 16–13
summary error report of device statistics, 16–12
suspending and resuming under SJ monitor, 16–5
uses for, 16–1
Error logging subsystem
description of, 16–2
Errors
during copy operations
ignoring, 13–13
ERROUT
function of, 16–3
options
and keyboard command equivalents (table), B–3
Executable files
creating at link time, 11–1

/F
DIR option, 4–5
DUP option, 6–8
LINK option, 11–47
PIP option, 13–12
File exchange program
See FILEX
Files
backing up with BUP, 3–3
changing volumes while manipulating, 13–10
concatenating, 13–16
copying
between RT–11 and DOS/BATCH or RSTS, 7–4
between RT–11 and interchange diskette, 7–6
from DECsystem–10 to RT–11, 7–8
in image mode with BATCH, A–15
many to several output volumes, 13–17
on a double-drive system, 13–11
on a single-drive system, 13–11
one block at a time, 13–16
those created before specified date, 13–14
those created on or after specified date, 13–14
those with certain date, 13–9
creating, 6–2
with BATCH, A–16
creating several copies of, 13–14
deleting, 13–9
after printing, 17–5
.BAD, 13–9
before copy, 13–14
.SYS, 13–9
with BATCH, A–18
excluding from an operation, 13–15
extending
with DUP, 6–12
with SIPP, 20–12
ignoring input errors when copying, 13–13
listing
See DIRECTORY utility program
preventing replacement of, 13–14
printing
multiple copies of, 17–6
those created before specified date, 17–6
those created since specified date, 17–6
those with specified date, 17–4
with banner pages, 17–5
with BATCH, A–31
without banner pages, 17–7
protecting, 13–12
recovering ones that are deleted, 6–4
renaming, 13–15
setting creation date for, 13–16
storing with BUP, 3–1
unprotecting, 13–18
File specifications
syntax of, 1–1
File types
and interchange format, 7–7
FILEX, 7–1 to 7–13
ASCII transfer, 7–3
calling, 7–2
changing volumes during operations, 7–12
device supported by (table), 7–1
function of, 7–1
halting, 7–2
image mode transfer, 7–3
options, 7–3 to 7–12
and keyboard command equivalents (table), B–3
options (table), 7–3
packed image mode transfer, 7–3
printing or changing volume ID with, 7–12
volume initialization, 7–11
suppressing confirmation message during, 7–11

Foreground job
  debugging with ODT, 18–21
  displaying status of, 14–3
FORLIB.OBJ
  including in a link, 11–47
FORMAT, 8–1 to 8–7
  calling, 8–1
  changing volumes during formatting or verifying, 8–7
  command string syntax, 8–2
  confirmation prompts, 8–2
  function of, 8–1
  halting, 8–1
  options, 8–3 to 8–7
    and keyboard command equivalents (table), B–3
  options (table), 8–3
  pattern verification, 8–4
  pattern verification (table), 8–5
  suppressing confirmation prompts, 8–7
  using while a foreground job is loaded, 8–2
$FORTRAN
  BATCH command, A–21
FORTRAN OTS
  processing by LINK, 11–15

/G
  DIR option, 4–6
  LIBR option, 10–8
  LINK option, 11–48
  PIP option, 13–13
GBL
  p-sect attribute, 11–5
Global sections, 11–4
Global symbols
  creating, 11–7
  definition of, 11–2
  forcing inclusion of during link, 11–49
  function of, 11–7
  listing in alphabetical order, 11–43
  processing of by LINK, 11–7
  referencing, 11–7
  resolution of, 11–7

/H
  BINCOM option, 2–3
  LINK option, 11–48
  PIP option, 13–13
  QUEMAN option, 17–5
  RESORC option, 14–5
Hardware configuration
  displaying, 14–2, 14–5, 14–10
High limit
  definition of, 11–19

/I
  BUP option, 3–2, 3–4
  DUP option, 6–5
  FILEX option, 7–3
  LINK option, 11–49
  PIP option, 13–14
  QUEMAN option, 17–6
I
  p-sect attribute, 11–5
Image mode copy, 13–3, 13–8
  for volumes, 6–5
Indirect command files
  nesting depth
    displaying allowed, 14–3
  running SIPP from, 20–16
Initializing volumes, 6–16
  with FILEX, 7–11
Interchange diskettes, 7–6
  copying files to and from, 7–6, 7–7

/J
  DIR option, 4–6
  PIP option, 13–14
  QUEMAN option, 17–6
  RESORC option, 14–5
$JOB
  BATCH command, A–23
Jobs
  loaded
    displaying status of, 14–5
Job status
  displaying, 14–3

/K
  DIR option, 4–6
  DUP option, 6–7
  LINK option, 11–49
  PIP option, 13–14
  QUEMAN option, 17–6

/L
  BUP option, 3–2, 3–6
  DIR option, 4–7
  FILEX option, 7–9
  LD option, 9–3
  LINK option, 11–49
  MACRO option, 12–4
  QUEMAN option, 17–6
  RESORC option, 14–6
  SIPP option, 20–3
  SLP option, 21–3
  SRCCOM option, 15–6
LCL
  p-sect attribute, 11–5

LD, 9–1 to 9–4
calling, 9–1
command string syntax, 9–2
options, 9–2, 9–2 to 9–4
and keyboard command equivalents (table), B–3
options (table), 9–2
terminating, 9–1
.LDA files
generating, 11–16, 11–49
LIBR, 10–1 to 10–15
calling, 10–1
command string syntax, 10–2
continuing command lines, 10–6
default file types, 10–2
function of, 10–1
halting, 10–1
options, 10–4, 10–5 to 10–11
and keyboard command equivalents (table), B–3
combining, 10–13
for macro libraries, 10–13, 10–14
for macro libraries (table), 10–14
options (table), 10–5
Librarian utility program
See LIBR
$LIBRARY
BATCH command, A–24
Library files
accessing object modules within, 10–1
as input to LINK, 11–12
copying, 13–9
definition of, 10–1, 11–2
directory of
including all global symbols, 10–5
wide, 10–11
including in link operations using BATCH, A–24
including module names in directory of, 10–8
including p-sect names in directory of, 10–9
macro
creating, 10–14
designating in MACRO command, 12–8
multiple definition, 11–15
object
creating, 10–3
deleting global symbols from the directory of, 10–8
deleting modules from, 10–6
extracting modules from, 10–7
inserting modules into, 10–3
merging, 10–4
obtaining directory listings of, 10–11
replacing modules in, 10–9
updating, 10–10
processing by LINK, 11–13
with multiply defined global entry points, 10–11
Library modules
calling other library modules, 11–13
definition of, 11–2
duplicating in overlay segments, 11–45
processing by LINK, 11–12, 11–13
Library routine list
changing amount of space allocated for, 11–52
.LIMIT special features
enabling, 11–55
$LINK
BATCH command, A–25
LINK, 11–1 to 11–57
calling, 11–8
command line
continuing, 11–44
command string syntax, 11–8
default file specifications, 11–9
function of, 11–1, 11–2
input library modules for, 11–12
input object modules for, 11–12
object module processing, 11–12
options, 11–22, 11–33, 11–43 to 11–56
and keyboard command equivalents (table), B–4
options (table), 11–10
output load module, 11–16
processing, 11–3
description of, 11–2
prompts, 11–57
sequence of, 11–57
symbol table
allowing largest possible area for, 11–53
terminating, 11–8
Link operations
using BATCH, A–25
Load maps
contents of, 11–19
description of, 11–18
sample, 11–18
with unmapped and virtual overlays, 11–38
wide
creating, 11–55

Load modules
arrangment of, 11–16
as LINK output, 11–16
creation of, 11–3
definition of, 11–2
relocatable code
specifying lowest address to be used by, 11–43
specifying a value to fill unused locations in, 11–56
Local sections, 11–4
Logical device names
assigning
to logical disks, 9–2
with BATCH, A–30
deassigning with BATCH, A–19
displaying assignments of, 14–6
using with BATCH, A–6
Logical disks, 9–1
See also LD
assigning to files, 9–3
freeing from file assignment, 9–4
uses for, 9–1
verifying and correcting assignments, 9–3
write-enabling, 9–4
write-locking, 9–4
Logical disk subsetting
displaying assignments for, 14–3, 14–8
Log of files
requesting when copying files, 13–17
requesting when printing files, 17–9
Low memory
definition, 11–2

/M
DIR option, 4–7
LIBR option, 10–14
LINK option, 11–50
MACRO option, 12–8
PIP option, 13–5
QUEMAN option, 17–7
RESORC option, 14–7
$MACRO
BATCH command, A–26
MACRO
options
and keyboard command equivalents (table), B–4
MACRO assembler, 12–1 to 12–13
calling, 12–1
calling using BATCH, A–26
command string syntax, 12–2
default file specifications (table), 12–3
error codes, 12–12
function control options, 12–6
listing control options, 12–4
options, 12–4 to 12–8
options (table), 12–4
output from, 12–1
temporary work file
assigning, 12–3
terminating, 12–3
Macro library files
creating, 10–14
Magtapes
copying from, 13–6
copying to or from with DUP, 6–9
dumping, 5–2
in BUP operations, 3–2
initializing for use with BUP, 3–2, 3–3
TSV05
using at 100ips during BUP operations, 3–2
using with PIP, 13–5
writing to, 13–7
Memory
amount on system
displaying, 14–5
limiting amount allocated by .SETTOP, 11–49
organization of
displaying, 14–3, 14–9
Memory image files
See .SAV files
Memory locations
modifying with LINK, 11–16
Memory usage bitmap
See Bitmap
$MESSAGE
BATCH command, A–29
Messages
sending to the console with BATCH, A–29
Monitors
bootstrapping with DUP, 6–9
Monitor type and version
displaying, 14–2, 14–7
$MOUNT
BATCH command, A–30
Multiple definition libraries
creating, 10–11
enlarging LINK's directory buffer for, 11–48
processing by LINK, 11–15

/N
DIR option, 4–7
DUP option, 6–16
LIBR option, 10–8
LINK option, 11–50
MACRO option, 12–4
PIP option, 13–14
QUEMAN option, 17–7
SLP option, 21–3

/O
BINCOM option, 2–3
DIR option, 4–8
DUP option, 6–9
LINK option, 11–22, 11–51
PIP option, 13–14
RESORC option, 14–7
Object module patch program
See PAT
Object modules
adding subroutines to, 19–6
creating, 11–12
definition of, 11–2
LINK processing of, 11–12
replacing lines in, 19–5
updating with PAT, 19–2
See also PAT
ODT, 18–1 to 18–27
accessing general registers 0–7, 18–9
accessing program's internal registers, 18–10
ASCII terminators (table), 18–20
automatic relocation, 18–5
back-arrow command, 18–8
breakpoints, 18–12
removing, 18–12
using, 18–22
calculating offsets with, 18–17
changing ASCII text with, 18–20
changing contents of locations with, 18–7
closing locations with, 18–7
commands, 18–6
constant register, 18–16
debugging background jobs with, 18–21
debugging foreground jobs with, 18–21
effective address search, 18–16
error detection, 18–26
format of output, 18–6
function of, 18–1
halting, 18–5
initializing memory blocks with, 18–16
internal organization, 18–21
internal registers to access with, 18–10
line feed key command, 18–8
linking low in memory, 18–2
linking with a program, 18–1
linking with overlaid files, 18–2
opening addressed location with, 18–9
opening byte locations, 18–8
opening location at relative branch offset, 18–9
opening locations indexed by the program counter, 18–8
opening locations with, 18–7
opening next location with, 18–8
opening previous location with, 18–8
priority level register ($P), 18–19
r, 18–13
Radix–50 terminators (table), 18–11
relocation calculators n! and nR, 18–19
relocation register commands, 18–18
running a program with, 18–13
searching, 18–25
for bit patterns, 18–15
single-instruction mode, 18–14
single-instruction mode commands (table), 18–14
start and restart addresses, 18–2
terminal interrupt processing, 18–25
up-arrow command, 18–8
using Radix–50, 18–11
using with display hardware, 18–21
word search, 18–15
X command, 18–11
On-line debugging technique
See ODT
Overlay handler
extended memory, 11–41
low memory, 11–23
Overlay regions
calculating the size of, 11–22
definition of, 11–20
virtual, 11–32
number of, 11–31
Overlays
See Overlay segments
Overlay segments
definition of
calling, 11–22
calling (example), 11–25
combining low memory with extended memory, 11–36
definition of, 11–20
extended memory, 11–28
converting program to use, 11–28

creating, 11–33, 11–55
definition of, 11–20
extending with SIPP, 20–12
low memory
creating, 11–51
definition of, 11–20
description of, 11–20
preserving return path when calling, 11–22
Overlay structure
creating, 11–20, 11–22, 11–26
extended memory, 11–30
of a FORTRAN program (example), 11–21
OVR
p-sect attribute, 11–5

/P
DIR option, 4–8
FILEX option, 7–3
FORMAT option, 8–4
LIBR option, 10–9
LINK option, 11–52
PIP option, 13–15
QUEMAN option, 17–7
SLP option, 21–3
PAT, 19–1 to 19–8
adding a subroutine to a module with, 19–6
calling, 19–1
checksum, 19–2, 19–8
command string syntax, 19–2
correction file, 19–1
format of, 19–4
function of, 19–1
halting, 19–1
processing of duplicate PSECTs and CSECTs, 19–4
processing of new global symbols, 19–4
replacing module lines with, 19–5
updating an object module with, 19–2
updating an object module with (figure), 19–2
Peripheral interchange program
See PIP
Physical address space
with low memory overlays, 11–31
PIP, 13–1 to 13–19
ASCII mode copy, 13–8
binary mode copy, 13–9
calling, 13–1
command string syntax, 13–1
function of, 13–1
image mode copy, 13–3, 13–8
options, 13–5 to 13–18
and keyboard command equivalents (table), B–5
options (table), 13–4
terminating, 13–1
treatment of .BAD files, 13–3
treatment of .SYS files, 13–18
wildcards with, 13–2
$PRINT
BATCH command, A–31
Privileged jobs
address space available for, 11–33
Program sections
See P-sects
Program virtual address space
See PVAS
Prompts
system (table), 1–3
Protecting files from deletion, 13–12
.PSECT directive, 11–4
P-sects
absolute base address
specifying, 11–52
allocation of memory for, 11–5
attributes, 11–4
attributes (table), 11–5
contents of, 11–4
creating a, 11–4
definition of, 11–2
description of, 11–4
extending in the root, 11–47
ordering in memory, 11–6
ordering in memory (table), 11–6
specifying starting address of in the root, 11–56
PVAS
definition of, 11–30
illustration of, 11–30
structure of, 11–30

/Q
BINCOM option, 2–4
DIR option, 4–9
DUP option, 6–10
LINK option, 11–52
PIP option, 13–15
QUEMAN option, 17–8
QUEMAN, 17–2 to 17–10
calling, 17–2
command string syntax, 17–2
continuing on several lines, 17–10
options, 17–4 to 17–9
and keyboard command equivalents (table), B–5
options (table), 17–3

QUEUE, 17–1 to 17–10
calling, 17–2
resuming or restarting output from, 17–9
suspending output from, 17–8
terminating, 17–4
Queue
listing contents of, 17–6
removing a job from the, 17–7
Queue Package
function of, 17–1
QUFILE.WRK
Queue Package work file, 17–1
setting default for deletion of, 17–7

/R
DIR option, 4–9
DUP option, 6–17
LD option, 9–4
LIBR option, 10–9
LINK option, 11–53
PIP option, 13–15
QUEMAN option, 17–9
REL
p-sect attribute, 11–5
.REL file
calculating absolute addresses for, 11–19
creating, 11–16
creating for foreground job, 11–53
Relocatable code
in load module
specifying highest address to be used by, 11–48
Relocatable expressions, 18–6
Relocatable file
See .REL file
Relocatable load module
creating, 11–16
Relocation bias for object modules, 18–5
Reloc-code
p-sect attributes, 11–5
RESORC, 14–1 to 14–11
calling, 14–1
function of, 14–1
options, 14–2 to 14–10
and keyboard command equivalents (table), B–6
options (table), 14–2
terminating, 14–1
Resource utility program
See RESORC
Restoring BUP volumes and files, 3–2
RO
p-sect attribute, 11–5
Root
See Root segment
Root segment
definition of, 11–2
rounding up the size of, 11–55
RSTS volumes
copying files to and from, 7–4
$RT11
BATCH command, A–32
$RUN
BATCH command, A–32
RW
p-sect attribute, 11–5

/S
BINCOM option, 2–4
DIR option, 4–9
DUP option, 6–11
FILEX option, 7–4
FORMAT option, 8–6
LINK option, 11–53
PIP option, 13–16
QUEMAN option, 17–8
RESORC option, 14–8
SLP option, 21–3
SRCCOM option, 15–4
SAV
p-sect attribute, 11–5
Save image files
examining and modifying
See SIPP
Save image patch program
See SIPP
.SAV files
creating, 11–16
Scope-code
function of, 11–4
p-sect attributes, 11–5
$SEQUENCE
BATCH command, A–33
SET options in effect
displaying, 14–3, 14–10
.SETTOP programmed request
limiting amount of memory allocated by, 11–49
special features for enabling, 11–55
Single-density format
writing on double-density diskettes, 8–6
SIPP
advancing through files by bytes, 20–6

ASCII mode, 20–7
backing up by bytes through files with, 20–6
backing up by words through files with, 20–6
Base? prompt, 20–3
calling, 20–1
changing from word mode to byte mode, 20–6
checksum, 20–3, 20–15
commands (table), 20–4
command string syntax, 20–2
creating a command file with, 20–2
creating only a command file as output with, 20–3
dialog, 20–2
display, 20–4
extending files and overlay segments, 20–12
extending non-overlaid program, 20–13
extending program with extended memory overlays only, 20–13
extending program with low and extended memory overlays, 20–14
extending program with low memory overlays only, 20–13
file searches, 20–9
  ASCII, 20–9
  Radix–50, 20–9
function of, 20–1
halting, 20–2
input command file
  creating with BINCOM, 2–3 to 2–6
inserting ASCII values, 20–8
inserting Radix–50 characters, 20–8
making modifications permanent, 20–12
modifying only locations you specify, 20–3
octal mode, 20–7
Offset? prompt, 20–3
opening and modifying locations with, 20–6
options (table), 20–3
prompting for an overlaid file, 20–2
Radix–50 mode, 20–8
recalling previous dialog prompt, 20–11
running from a BATCH stream, 20–17
running from an indirect command file, 20–16
Segment? prompt, 20–2
suppressing modification of input file, 20–3
verifying changes made by, 20–10
SLP, 21–1 to 21–11
  adding lines to a source file with, 21–7
  audit trail, 21–5
    disabling, 21–5
    disabling generation of, 21–3
    enabling, 21–5
    specifying size of, 21–3
    specifying start column for, 21–3
  backup file
    suppressing, 21–3
  calling, 21–1
  checksum, 21–3, 21–11
  command file
    creating with SRCCOM, 15–8
  command string syntax, 21–1
  creating a double-spaced listing with, 21–3
  creating a numbered listing with, 21–6
  deleting lines from a source file with, 21–9
  formatting output file for, 21–3
  function of, 21–1
  halting, 21–1
  input command file
    creating using a text editor, 21–4
    creating with SRCCOM, 21–1
  options, 21–2
  options (table), 21–3
  replacing lines in a source file with, 21–10
  size of source line to, 21–3
  update commands (table), 21–5
  update line format, 21–4
  update text
    ending, 21–5
    starting, 21–5
Source comparison program
  See SRCCOM
Source files
  adding lines to, 21–7
  comparing
    See SRCCOM
  deleting lines from, 21–9
  modifying
    See SLP
  replacing lines in, 21–10
Source language patch program
  See SLP
Squeeze operation
  squeezing the system device, 6–11
  squeezing volumes, 6–11
SRCCOM, 15–1
  calling, 15–1
  command string syntax, 15–1

creating SLP command file with, 15–8
differences listing
format of, 15–3
including form feeds in, 15–4
interpreting, 15–4
halting, 15–1
ignoring comments during comparision, 15–4
ignoring spaces and tabs during comparision, 15–4
including blank lines in comparison, 15–7
including changebars and bullets in differences listing, 15–7
including trailing blanks and tabs during comparison, 15–4
options, 15–4 to 15–7
and keyboard command equivalents (table), B–6
options (table), 15–4
processing of files, 15–2
setting number of lines for matching, 15–6
specifying audit trail for SLP command file, 15–8
specifying number of lines to match for, 15–4
wildcards with, 15–2
Stack
definition of, 11–3
modifying address of the, 11–50
.STB file
as input to LINK, 11–12
Symbol table definition file
See .STB file
Symbol table overflow
correcting with LINK /P, 11–52
SYSCOM area
and LINK, 11–3
.SYS files
PIP treatment of, 13–3, 13–18
SYSLIB.OBJ
processing by LINK, 11–15
System communication area
See SYSCOM area
System device
displaying, 14–3
System generation options
displaying those in effect, 14–3, 14–7, 14–10

/T
DIR option, 4–11
FILEX option, 7–8
LINK option, 11–54
PIP option, 13–16
RESORC option, 14–8
SLP option, 21–3
SRCCOM option, 15–4
/T:n
DUP option, 6–12
Terminals
displaying status of, 14–3
Terminal status
displaying, 14–8
Transfer address
definition of, 11–19
specifying, 11–54
TSV05 magtape
100ips streaming during BUP operations, 3–2
Type code
p-sect attributes, 11–5

/U
DIR option, 4–11
DUP option, 6–13
FILEX option, 7–6
LIBR option, 10–10
LINK option, 11–55
PIP option, 13–16
Uninitializing a volume, 6–19
Unprotecting files, 13–18
Utility programs
calling, 1–1
list of, B–1
options
and keyboard command equivalents (table), B–1

/V
DIR option, 4–11
DUP option, 6–14, 6–17
FILEX option, 7–12
FORMAT option, 8–6
LINK option, 11–33, 11–55
PIP option, 13–17
SRCCOM option, 15–4
Verification
of copy operation, 13–13
of volumes, 8–4, 8–6
Virtual jobs
address space available for, 11–33
Volume formatting utility program
See FORMAT
Volume ID
changing, 6–17
printing or changing, 6–14
for magtape, 6–14
printing or changing with FILEX, 7–12

Volumes
backing up with BUP, 3–4
image copying, 6–5
initialized
restoring, 6–19
initializing, 6–16
squeezing, 6–11
storing with BUP, 3–1
verifying, 8–4, 8–6
VSECT attributes (table), 11–6

/W
DUP option, 6–15
FILEX option, 7–12
FORMAT option, 8–7
LD option, 9–4
LIBR option, 10–11
LINK option, 11–55
PIP option, 13–17
QUEMAN option, 17–9
Wildcards
BUP treatment of, 3–1
using
with BATCH, A–6
with PIP, 13–2
with SRCCOM, 15–2

/X
BUP option, 3–2, 3–7
LIBR option, 10–11
LINK option, 11–56
PIP option, 13–18
QUEMAN option, 17–9
RESORC option, 14–9

Y
BUP option, 3–2
DUP option, 6–15
FILEX option, 7–11
FORMAT option, 8–7
LINK option, 11–56
PIP option, 13–18

/Z
BUP option, 3–2, 3–8
DUP option, 6–16
FILEX option, 7–11
LINK option, 11–56
PIP option, 13–18
RESORC option, 14–10

HOW TO ORDER ADDITIONAL DOCUMENTATION

From	Call	Write
Chicago	312–640–5612 8:15 A.M. to 5:00 P.M. CT	Digital Equipment Corporation Accessories & Supplies Center 1050 East Remington Road Schaumburg, IL 60195
San Francisco	408–734–4915 8:15 A.M. to 5:00 P.M. PT	Digital Equipment Corporation Accessories & Supplies Center 632 Caribbean Drive Sunnyvale, CA 94086
Alaska, Hawaii	603–884–6660 8:30 A.M. to 6:00 P.M. ET *or* 408–734–4915 8:15 A.M. to 5:00 P.M. PT	
New Hampshire	603–884–6660 8:30 A.M. to 6:00 P.M. ET	Digital Equipment Corporation Accessories & Supplies Center P.O. Box CS2008 Nashua, NH 03061
Rest of U.S.A., Puerto Rico*	1–800–258–1710 8:30 A.M. to 6:00 P.M. ET	

*Prepaid orders from Puerto Rico must be placed with the local DIGITAL subsidiary (call 809–754–7575)

Canada British Columbia	1–800–267–6146 8:00 A.M. to 5:00 P.M. ET	Digital Equipment of Canada Ltd 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: A&SG Business Manager
Ottawa–Hull	613–234–7726 8:00 A.M. to 5:00 P.M. ET	
Elsewhere	112–800–267–6146 8:00 A.M. to 5:00 P.M. ET	
Elsewhere		Digital Equipment Corporation A&SG Business Manager*

*c/o DIGITAL's local subsidiary or approved distributor

RT–11 System
Utilities Manual
AA–M239A–TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

— Assembly language programmer
— Higher-level language programmer
— Occasional programmer (experienced)
— User with little programming experience
— Student programmer
— Other (please specify) _____

Name _____ Date _____

Organization _____ Telephone _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear — Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG/ML PUBLICATIONS, MLO5–5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MA 01754

Do Not Tear — Fold Here

Cut Along Dotted Line